a2\test.py

```python
import os # Import the os module for file operations.
import json # Import the json module for working with JSON data (history).
import pytest # Import the pytest module for automated testing.
from app import app  # Import the Flask app from app.py.
from unittest.mock import patch # Import the patch function from unittest.mock to mock requests.
import requests # Import the requests module for making HTTP requests.

@pytest.fixture
def client():
    #
    # Fixture Client
    # This fixture provides a test client for the Flask application to simulate HTTP requests during tests. This fixture is automatically invoked by pytest when a
    # test function includes `client` as an argument. It allows test functions to simulate GET, POST, or other HTTP requests to the Flask app.
    # The client object is available for interacting with the app's routes, and it will be cleaned up after the test.
    #
    with app.test_client() as client: # Create a test client for the Flask app using app.test_client(). The test client simulates HTTP requests to the Flask app.
        yield client # Yield the test client to the test functions. This allows the test functions to use the test client to make requests to the Flask app.

def test_index(client):
    #
    # Test Case 1: Testing the index of the aggregator. This will confirm that the index page is reachable and correct.
    # Execution: python -m pytest test.py -k "test_index" -s -v # Only use -s to view the messages in the test.
    # This method will test the index page of the Flask app. It sends a GET request to the root URL ('/') and checks if the response status code is 200 (OK) and if the
    # response data contains the text "Keyword Search Aggregator".
    # Expected Result: Pass. The page should load successfully and display the "Keyword Search Aggregator" text.
    #
    response = client.get('/') # Send a GET request to the root URL ('/') using the test client.
    assert response.status_code == 200 # Check if the response status code is 200 (OK).
    assert b"Keyword Search Aggregator" in response.data # Check if the response data contains the text "Keyword Search Aggregator".
    print("Index page loaded successfully") # Print a message indicating that the index page loaded successfully.

def test_search(client):
    #
    # Test Case 2: Testing the search functionality of the `/search` route with a query parameter.
    # Execution: python -m pytest test.py -k "test_search" -s -v # Use -s to see the printed search results in the console.
```

```
    # This method verifies that the search functionality works when the user submits a query. The test sends a GET request to the `/search` endpoint with the query parameter, checks that the status code is 200 (OK),
    # and prints the first 10 search result URLs from both Google Search and Google News. It also ensures that the response contains the expected "organic_results" and "news_results" for Google Search and Google News respectively.
    # Expected Result: Pass. The search functionality should return valid search results, and the results should contain the appropriate URLs.
    #
    keyword = "Champion's League" # Define a search keyword for testing.
    response = client.get(f'/search?q={keyword}') # Send a GET request to the `/search` endpoint with the search keyword using the test client.
    assert response.status_code == 200 # Check if the response status code is 200 (OK).
    data = response.json # Parse the response data as JSON.
    print("Search Results:") # Print a message indicating that the search results are being displayed.
    for result in data["google_search"].get("organic_results", [])[:10]:  # Loop through the first 10 search results from Google Search.
        print(f"Google Search - URL: {result.get('link')}") # Print the URL of each search result from Google Search.
    for result in data["google_news"].get("news_results", [])[:10]: # Loop through the first 10 search results from Google News.
        print(f"Google News - URL: {result.get('link')}") # Print the URL of each search result from Google News.
    assert "organic_results" in data["google_search"] # Check if the response data contains the key "organic_results" for Google Search.
    assert "news_results" in data["google_news"] # Check if the response data contains the key "news_results" for Google News.

def test_MissingQuery(client):
    #
    # Test Case 3: Testing the aggregator when the query parameter is missing from the `/search` route.
    # Execution: python -m pytest test.py -k "test_MissingQuery" -s -v # Use -s to view the error message printed in the console.
    # This method verifies that the `/search` route returns a 400 status code and the appropriate error message when no query parameter is provided.
    # The test sends a GET request to the `/search` endpoint without the query parameter and checks that the response status code is 400 (Bad Request).
    # It also ensures that the response contains the correct error message indicating that the query parameter 'q' is missing.
    # Expected Result: Pass. The route should return a 400 status code and an error message stating that the 'q' parameter is missing.
    #
    response = client.get('/search') # Send a GET request to the `/search` endpoint without the query parameter 'q' using the test client.
    assert response.status_code == 400 # Check if the response status code is 400 (Bad Request).
    assert response.json['error'] == 'Missing query parameter "q"' # Check if the response contains the expected error message from app.py.
    print("Error: Missing query parameter") # Print a message indicating that the query parameter is missing.

```

```python
def test_InvalidSearch(client):
    #
    # Test Case 4: Testing the `/search` route with an invalid keyword ("!!!").
    # Execution: python -m pytest test.py -k "test_InvalidSearch" -s -v # Use -s to see printed results in the console.
    # This method verifies that when an invalid keyword "!!!" is provided, the API returns no search results.
    # It checks that both the Google Search and Google News sections are empty in the response.
    # Expected Result: Pass. The search results should be empty for both Google Search and Google News.
    #
    invalid_keyword = "!!!"  # Define an invalid search keyword.
    response = client.get(f'/search?q={invalid_keyword}')  # Send a GET request to the `/search` endpoint with the invalid keyword.
    assert response.status_code == 200 # Check if the response status code is 200 (OK).
    data = response.json  # Parse the response data as JSON.
    print(data) # Print the response data for debugging purposes. There will be no valid URLs.
    assert "google_search" in data  # Ensure the 'google_search' key exists.
    assert len(data["google_search"].get("organic_results", [])) == 0  # Ensure no valid search results from google_search.
    assert "google_news" in data  # Ensure the 'google_news' key exists.
    assert len(data["google_news"].get("news_results", [])) == 0  # Ensure no valid search results from google_news.
    print("No search results found") # Print a message indicating that no search results were found.

def test_history(client):
    #
    # Test Case 5: Testing the history functionality of the `/history` route to retrieve stored search history.
    # Execution: python -m pytest test.py -k "test_history" -s -v # Use -s to see the printed search history in the console.
    # This method verifies that the history functionality works as expected. It sends a GET request to the `/history` route, checks that the response status code is 200 (OK),
    # and prints the search history entries along with the associated URLs and sources. It also ensures that the response is a list, even if it is empty.
    # Expected Result: Pass. The history functionality should return valid search history entries, and the response should be a list of records with keywords and results.
    #
    response = client.get('/history') # Send a GET request to the `/history` endpoint using the test client.
    assert response.status_code == 200 # Check if the response status code is 200 (OK).
    history = response.json # Parse the response data as JSON to retrieve the search history.
    assert isinstance(history, list) # Check that the search history is a list (even if empty).
    print("Search History:") # Print a message indicating that the search history is being displayed.
    for entry in history: # Loop through each search history entry.
        print(f"Keyword: {entry['keyword']}") # Print the keyword associated with the search history entry.
```

```
        for link in entry['results']: # Loop through the search result links in the entry.
            print(f"  Source: {link['source']} - URL: {link['link']}") # Print the source and URL of each search result link.

def test_SearchHistoryFileHandling(client):
    #
    # Test Case 6: Testing the creation and updating of `search_history.json` file.
    # Execution: python -m pytest test.py -k "test_SearchHistoryFileHandling" -s -v # Use -s to see printed results in the console.
    # This test ensures that when a search is performed and `search_history.json` does not exist, it is created. Additionally, it checks that the file is updated properly when new searches are performed.
    # Expected Result: Pass. The file should be created if it doesn't exist and updated with the new search history when new searches are added.
    #
    SEARCH_HISTORY_FILE = 'search_history.json' # Define the search history file name.

    if os.path.exists(SEARCH_HISTORY_FILE): # Check if the search history file exists.
        os.remove(SEARCH_HISTORY_FILE) # Remove the search history file if it exists (for test purposes).
    print(f"Initial check: Does {SEARCH_HISTORY_FILE} exist? {os.path.exists(SEARCH_HISTORY_FILE)}\n") # Print a message to state if the file exists initially.
    response = client.get('/search?q=Canada') # Simulate a search to check if the history file is created.
    assert response.status_code == 200  # Check if the response status code is 200 (OK).
    assert os.path.exists(SEARCH_HISTORY_FILE), "search_history.json file should be created." # Check if the search history file has been created.
    print(f"After first search: Does {SEARCH_HISTORY_FILE} exist? {os.path.exists(SEARCH_HISTORY_FILE)}\n") # Print a message to state if the file exists after the first search.

    with open(SEARCH_HISTORY_FILE, 'r') as file: # Open the search history file to read its contents.
        history = json.load(file) # Load the JSON data from the file.
    print(f"After first search, history content: {history}\n") # Print the contents of the search history file after the first search.
    assert isinstance(history, list), "Search history should be a list." # Check if the search history is a list.
    assert len(history) == 1, "There should be one entry in the search history." # Check if there is one entry in the search history.
    assert history[0]['keyword'] == 'Canada', "The first entry should match the search keyword." # Check if the first entry in the search history matches the search keyword.

    response = client.get('/search?q=America')  # Perform a second search to check if the history file is updated.
    assert response.status_code == 200  # Check if the response status code is 200 (OK).
    print(f"After second search: Does {SEARCH_HISTORY_FILE} exist? {os.path.exists(SEARCH_HISTORY_FILE)}\n") # Print a message to state if the file exists after the second search.
    with open(SEARCH_HISTORY_FILE, 'r') as file: # Open the search history file to read its contents after the second search.
```

```
131         history = json.load(file) # Load the JSON data from the file.
132     print(f"After second search, history content: {history}\n") # Print the contents of the search history file after the second search.
133     assert isinstance(history, list), "Search history should be a list."  # Check if the search history is a list.
134     assert len(history) == 2, "There should be two entries in the search history." # Check if there are two entries in the search history.
135     assert history[0]['keyword'] == 'America', "The first entry should be the second search keyword." # Check if the first entry in the search history matches the second search keyword.
136     assert history[1]['keyword'] == 'Canada', "The second entry should be the first search keyword." # Check if the second entry in the search history matches the first search keyword.
137     print("search_history.json file is being created and updated properly.") # Print a message indicating that the search history file is being created and updated correctly.
138
139 def test_APIFailure(client):
140     #
141     # Test Case 7: Testing the `/search` route when the SerpAPI fails to respond.
142     # Execution: python -m pytest test.py -k "test_APIFailure" -s -v # Use -s to see printed results in the console.
143     # This method verifies that when the SerpAPI service fails, the `/search` endpoint handles the failure gracefully by returning a 500 status code and an
144     # appropriate error message. It uses a mock request to simulate a failure when contacting the SerpAPI service.
145     # Expected Result: Pass. The `/search` route should return a 500 status code and an error message indicating the failure to contact the SerpAPI or another internal error.
146     #
147     with patch('requests.get') as mock_get: # Patch the requests.get method to simulate a failure when contacting the SerpAPI.
148         mock_get.side_effect = requests.RequestException("SerpAPI is down") # Raise an exception to simulate the SerpAPI service being down.
149         response = client.get('/search?q=Test') # Send a GET request to the `/search` endpoint with a test query.
150         assert response.status_code == 500 # Check if the response status code is 500 (Internal Server Error).
151         assert 'error' in response.json # Check if the response contains an 'error' key.
152         assert response.json['error'] == "Error contacting SerpAPI: SerpAPI is down" # Ensure the error message matches what app.py generates.
153         print("SerpAPI is unreachable at this time") # Print a message indicating that the SerpAPI is unreachable.
```