Testing Report

Title: Report on Testing the API of the Game of Thrones Character Website

Date: September 11, 2023

General Summary

The report provides a detailed description of the process of testing the "Game of Thrones API v2" using the Swagger documentation available at https://thronesapi.com/swagger/index.html?urls.primaryName=Game%20of%20Thrones%20API%20v2. The primary objective of this testing was to ascertain the consistency and correctness of the data accessible through the API, as well as the identification of any deficiencies or errors in its implementation.

The report consists of a series of tests that cover various aspects of the API, including retrieving character data, data consistency based on ID search, character affiliations with families, accuracy of full character names, checking photo pathways, and modifying character data by sending POST requests.

Testing the Status Code of GET Requests and Displaying Character Data

```python
# test the status code of a GET request
def test_get_request_status_code():
    response = requests.get("https://thronesapi.com/api/v2/Characters")
    if response.status_code == 200:
        print("Success! Status code is 200")
    else:
        print(f"Error! Status code is {response.status_code}")

# display character profiles
def character_profiles():
    response = requests.get("https://thronesapi.com/api/v2/Characters").json()
    for character in response:
        character_profile = (
            "ID:", character["id"],
            "FIRST NAME:", character["firstName"],
            "LAST NAME:", character["lastName"],
            "FULL NAME:", character["fullName"],
            "TITLE:", character["title"],
            "FAMILY:", character["family"],
            "IMAGE:", character["image"],
            "IMAGE URL:", character["imageUrl"]
        )
        print(character_profile)
```

Screenshot 1:

Creating a Function to Send GET Requests and Display Character Data

- *Description:* The screenshot shows function definitions for sending GET requests and displaying character profile data.

The status code of the GET request is 200; the test has passed.

Testing Character Data Consistency Based on ID Search

```python
# test data consistency based on character ID lookup
def test_data_consistency():
    characters_list = requests.get("https://thronesapi.com/api/v2/Characters").json()
    characters = {}

    for character in characters_list:
        character_id = character["id"]
        characters[character_id] = character

    for id in characters.keys():
        character_data = requests.get(f"https://thronesapi.com/api/v2/Characters/{id}").json()

        if character_data != characters[id]:
            print(f"Inconsistency in data for character with ID {id}")

    print("Data consistency check completed.")
```

Screenshot 2:

Creating a Function to Verify Data Consistency Through ID-based Character Searches

- *Description:* The screenshot displays the definition of a function for testing the consistency of character data by comparing data obtained through the API for all characters collectively and data for individual characters based on ID search.

Data for all characters is consistent; the test has passed.

Testing Character Data

```python
# test character data
def test_character_data():
    character_data = requests.get("https://thronesapi.com/api/v2/Characters").json()

    for character_info in character_data:
        missing_data = []

        if "id" not in character_info or character_info["id"] is None:
            missing_data.append("ID")
        if not character_info.get("firstName"):
            missing_data.append("First Name")
        if not character_info.get("lastName"):
            missing_data.append("Last Name")
        if not character_info.get("fullName"):
            missing_data.append("Full Name")
        if not character_info.get("title"):
            missing_data.append("Title")
        if not character_info.get("family"):
            missing_data.append("Family")
        if not character_info.get("image"):
            missing_data.append("Image")
        if not character_info.get("imageUrl"):
            missing_data.append("Image URL")

        if missing_data:
            print(f"Character {character_info['id']} is missing data for: {', '.join(missing_data)}")
        else:
            print(f"Character {character_info['id']} has all data.")
```

```
Character 0 has all data.
Character 1 has all data.
Character 2 has all data.
Character 3 has all data.
Character 4 has all data.
Character 5 has all data.
Character 6 has all data.
Character 7 has all data.
Character 8 has all data.
Character 9 has all data.
Character 10 has all data.
Character 11 has all data.
Character 12 has all data.
Character 13 has all data.
Character 14 has all data.
Character 15 has all data.
Character 16 has all data.
Character 17 has all data.
Character 18 has all data.
Character 19 has all data.
Character 20 has all data.
Character 21 has all data.
Character 22 has all data.
Character 23 has all data.
Character 24 has all data.
Character 25 has all data.
Character 26 has all data.
Character 27 has all data.
Character 28 has all data.
Character 29 has all data.
Character 30 is missing data for: Last Name
Character 31 has all data.
Character 32 has all data.
Character 33 has all data.
Character 34 has all data.
Character 35 has all data.
Character 36 has all data.
Character 37 has all data.
Character 38 has all data.
Character 39 has all data.
Character 40 has all data.
Character 41 has all data.
Character 42 has all data.
Character 43 has all data.
Character 44 has all data.
Character 45 has all data.
Character 46 has all data.
Character 47 has all data.
Character 48 is missing data for: First Name, Family
Character 49 has all data.
Character 50 has all data.
Character 51 has all data.
Character 52 has all data.
```

Screenshot 3:

Creating a Function for Character Data Testing

- *Description:* The screenshot displays the definition of a function for testing character data, which outputs if any specific character data is missing.

Screenshot 4:

The Outcome of Executing the Character Data Testing Function

- *Description:* The screenshot shows the result of running the character data testing function. The character with ID 30 is missing the last name data, while the character with ID 48 is missing the first name and family data.

The test did not pass.

Testing Full Character Names

```python
# test character full names
def test_character_full_names():
    response = requests.get("https://thronesapi.com/api/v2/Characters").json()

    for character in response:
        first_name = character["firstName"].strip()
        last_name = character["lastName"].strip()
        full_name = character["fullName"].strip()

        if first_name and last_name and full_name != f"{first_name} {last_name}":
            print(f"ID: {character['id']}")
            print(f"First Name: {first_name}")
            print(f"Last Name: {last_name}")
            print(f"Full Name: {full_name}")
            print("Status: INCORRECT")
            print("-" * 30)
```

Screenshot 5:

Creating a Function for Full Character Name Testing

- *Description:* The screenshot displays the definition of a function for testing full character names, which is based on API documentation and constructs the full name of a character by combining their first name and last name.

The output is:

```
ID: 10
First Name: Cateyln
Last Name: Stark
Full Name: Catelyn Stark
Status: INCORRECT
------------------------------
ID: 11
First Name: Robb
Last Name: Stark
Full Name: Rob Stark
Status: INCORRECT
------------------------------
ID: 15
First Name: Sandor
Last Name: Clegane
Full Name: The Hound
Status: INCORRECT
------------------------------
ID: 19
First Name: Varys
Last Name: Unknown
Full Name: Varys
Status: INCORRECT
------------------------------
ID: 22
First Name: Ygritte
Last Name: None
Full Name: Ygritte
Status: INCORRECT
------------------------------
```

ID: 23
First Name: Brienne
Last Name: Tarth
Full Name: Brienne of Tarth
Status: INCORRECT

ID: 24
First Name: Missandei
Last Name: None
Full Name: Missandei
Status: INCORRECT

ID: 25
First Name: Gilly
Last Name: None
Full Name: Gilly
Status: INCORRECT

ID: 26
First Name: Viserys
Last Name: Targaryan
Full Name: Viserys Targaryn
Status: INCORRECT

ID: 29
First Name: Daario
Last Name: Naharis
Full Name: Daario
Status: INCORRECT

ID: 40
First Name: Melisandre
Last Name: The Red Woman
Full Name: Melisandre
Status: INCORRECT

ID: 47
First Name: Wylis
Last Name: Hodor
Full Name: Hodor
Status: INCORRECT

ID: 51
First Name: Qyburn
Last Name: Grand Maester
Full Name: Qyburn
Status: INCORRECT

The test did not pass.

Testing Character Family Membership Data

```python
# define excluded families to be ignored
EXCLUDED_FAMILIES = {"None", "Unknown"}

# list of correctly spelled family names
CORRECT_FAMILIES = {
    "Baelish", "Baratheon", "Bolton", "Bronn", "Clegane", "Free Folk", "Greyjoy",
    "Lannister", "Lorath", "Mormont", "Naathi", "Naharis", "Oyburn", "Sand", "Seaworth",
    "Sparrow", "Stark", "Targaryen", "Tarly", "Tarth", "Tyrell", "Viper", "Worm"
}
```

```python
# helper function to format family names
def format_family_name(family):
    formatted_family = family.replace("House", "").strip()
    return formatted_family

# finding the closest correctly spelled family from the CORRECT_FAMILIES list
def find_closest_family(correct_family_name, CORRECRT_FAMILIES, EXCLUDED_FAMILIES):
    # the difflib library is used to find the closest match based on similarity.
    # the 'n' parameter is set to 1 to retrieve only the closest match.
    # the 'cutoff' parameter is set to 0.8, meaning that only matches with a similarity of 80% or higher are considered.
    closest_match = difflib.get_close_matches(correct_family_name, CORRECT_FAMILIES.union(EXCLUDED_FAMILIES), n=1, cutoff=0.8)

    if closest_match:
        return closest_match[0]
    else:
        # return the original family name if no close match is found
        return correct_family_name
```

```python
def main():
    # fetch character data from the Thrones API
    response = requests.get("https://thronesapi.com/api/v2/Characters").json()

    # initialize counters and sets to track families
    family_counter = Counter()  # counter for counting occurrences of each family
    unique_families = set()     # set for storing unique formatted family names
    num_excluded_families = 0   # number of characters with excluded families
    characters_without_family = 0  # number of characters without a family

    # iterate through characters and analyze their family ties
    for character in response:
        family = character.get("family")
        if family and family.strip() != "":
            formatted_family = format_family_name(family)
            closest_correct_family = find_closest_family(formatted_family, CORRECT_FAMILIES, EXCLUDED_FAMILIES)

            if closest_correct_family:
                formatted_family_str = str(closest_correct_family)
                character["family"] = formatted_family_str

                if formatted_family_str not in EXCLUDED_FAMILIES:
                    family_counter[formatted_family_str] += 1
                    unique_families.add(formatted_family_str)
                else:
                    num_excluded_families += 1
            else:
                characters_without_family += 1
        else:
            characters_without_family += 1

    # helper function to print section headers
    def print_section_header(title):
        print("\n" + "=" * len(title))
        print(title)
        print("=" * len(title))

    # print section with a list of unique families and their statistics
    print_section_header("Families:")
    for index, family in enumerate(unique_families, start=1):
        print(f"{index}. {family}: {family_counter[family]}")

    # print section with character statistics
    print_section_header("Character Statistics:")
    total_characters = len(response)
    total_characters_without_family_or_excluded = characters_without_family + num_excluded_families
    total_characters_with_family = total_characters - total_characters_without_family_or_excluded

    print(f"Total characters: {total_characters}")
    print(f"Characters without a family: {characters_without_family}")
    print(f"Characters with excluded families: {num_excluded_families}")
    print(f"Total characters without a family or with excluded families: {total_characters_without_family_or_excluded}")
    print(f"Total characters with a family (excluding excluded families): {total_characters_with_family}")

    # print section with characters' family affiliations
    print_section_header("Character Family Affiliations:")
    for character in response:
        family = character.get("family")
        if family and family.strip() != "" and family not in EXCLUDED_FAMILIES:
            formatted_family = format_family_name(family)
            character["family"] = formatted_family
            if formatted_family in unique_families:
                print(f"{character['fullName']} belongs to the {formatted_family} family")
            else:
                print(f"{character['fullName']} has no family affiliation")
        else:
            print(f"{character['fullName']} has no family affiliation")
```

Screenshot 6:

Definition of Excluded and Correctly Spelled Families

- *Description:* The screenshot displays the definition of excluded families (family names that need to be excluded from the search due to inconsistent definitions when there is no available family data) and correctly spelled family names (family names without spelling errors).

Screenshot 7:

Definition of Auxiliary Functions for Formatting and Correcting Spelling Errors in Family Names

- *Description:* The screenshot shows the definition of auxiliary functions for formatting (a function used due to inconsistent definitions of known families, as is the case with "House Stark") and correcting spelling errors in family names.

Screenshot 8:

Definition of the Main Function for Testing Character Family Belonging

- *Description:* The screenshot displays the definition of the main function for testing character family belonging. This function includes the initialization of a counter and a set to prevent duplicate results. Subsequently, spelling errors are corrected and all family formats without data are extracted from the results, while known families are grouped into a unique format. The output includes a list of all families with statistics on character belonging. Additionally, the function includes an auxiliary function that highlights result headings for better readability.

The output is:

```
=========
Families:
=========
1. Targaryen: 3
2. Seaworth: 1
3. Lorath: 2
4. Baelish: 1
5. Free Folk: 2
6. Greyjoy: 3
7. Lannister: 5
8. Tarly: 1
9. Naharis: 1
10. Clegane: 1
11. Bronn: 1
12. Viper: 1
13. Stark: 10
14. Sand: 1
15. Mormont: 2
16. Bolton: 2
17. Naathi: 1
18. Sparrow: 1
19. Qyburn: 1
20. Worm: 1
21. Baratheon: 5
22. Tarth: 1
23. Tyrell: 2

====================
Character Statistics:
====================
Total characters: 53
Characters without a family: 1
Characters with excluded families: 3
Total characters without a family or with excluded families: 4
Total characters with a family (excluding excluded families): 49

============================
Character Family Affiliations:
============================
Daenerys Targaryen belongs to the Targaryen family
Samwell Tarly belongs to the Tarly family
Jon Snow belongs to the Stark family
Arya Stark belongs to the Stark family
Sansa Stark belongs to the Stark family
Brandon Stark belongs to the Stark family
Ned Stark belongs to the Stark family
Robert Baratheon belongs to the Baratheon family
Jamie Lannister belongs to the Lannister family
Cersei Lannister belongs to the Lannister family
Catelyn Stark belongs to the Stark family
Rob Stark belongs to the Stark family
Theon Greyjoy belongs to the Greyjoy family
Joffrey Baratheon belongs to the Lannister family
Tyrion Lannister belongs to the Lannister family
The Hound belongs to the Clegane family
Petyr Baelish belongs to the Baelish family
Davos Seaworth belongs to the Seaworth family
Stannis Baratheon belongs to the Baratheon family
Varys has no family affiliation
Khal Drogo belongs to the Targaryen family
```

```
Margaery Tyrell belongs to the Tyrell family
Ygritte belongs to the Free Folk family
Brienne of Tarth belongs to the Tarth family
Missandei belongs to the Naathi family
Gilly has no family affiliation
Viserys Targaryn belongs to the Targaryen family
Rickon Stark belongs to the Stark family
Roose Bolton belongs to the Bolton family
Daario belongs to the Naharis family
Shae belongs to the Lorath family
Tommen Baratheon belongs to the Baratheon family
Gendry Baratheon belongs to the Baratheon family
Jorah Mormont belongs to the Mormont family
Robert Baratheon belongs to the Baratheon family
Ramsey Bolton belongs to the Bolton family
Talisa Stark belongs to the Stark family
Jeor Mormont belongs to the Mormont family
The High Sparrow belongs to the Sparrow family
Oberyn Martell belongs to the Viper family
Melisandre has no family affiliation
Jaqen H'ghar belongs to the Lorath family
Tywin Lannister belongs to the Lannister family
Ellaria Sand belongs to the Sand family
Tormund Giantsbane belongs to the Free Folk family
Yara Greyjoy belongs to the Greyjoy family
Euron Greyjoy belongs to the Greyjoy family
Hodor belongs to the Stark family
Pycelle has no family affiliation
Grey Worm belongs to the Worm family
Olenna Tyrell belongs to the Tyrell family
Qyburn belongs to the Qyburn family
Lord Bronn belongs to the Bronn family
```

The test did not pass.

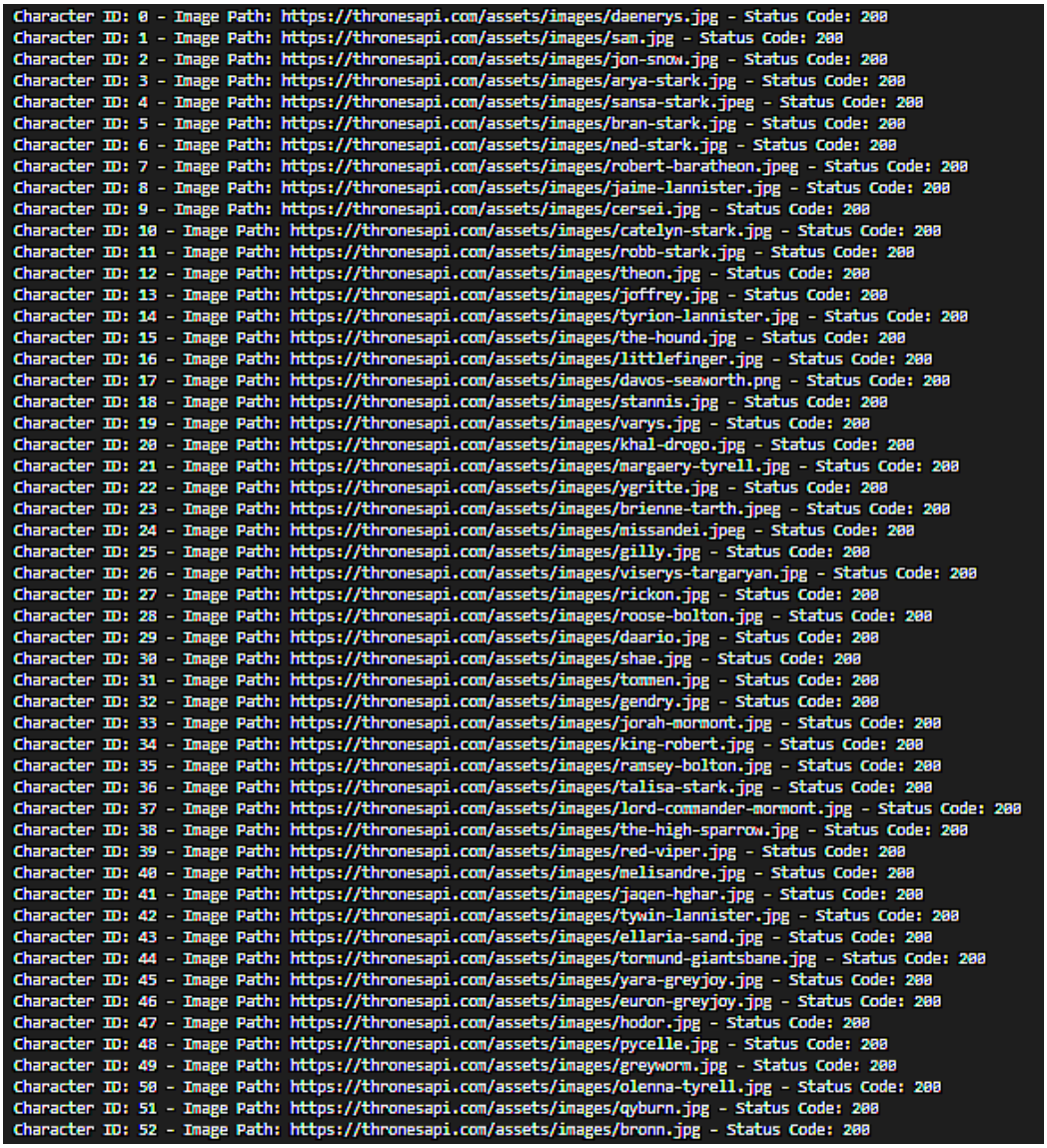
Testing Photo Path Verification

```python
# test image paths
def test_image_paths():
    # fetch character data from the API
    character_data = requests.get("https://thronesapi.com/api/v2/Characters").json()

    for character_info in character_data:
        character_id = character_info["id"]
        image_path = character_info["imageUrl"]

        # check if the image path exists before sending a request
        if requests.head(image_path).status_code == 200:
            response = requests.get(image_path)

            if response.status_code == 200:
                print(f"Character ID: {character_id} - Image Path: {image_path} - Status Code: {response.status_code}")
            else:
                print(f"Character ID: {character_id} - Image Path: {image_path} - Status Code: {response.status_code} (Error)")
        else:
            print(f"Character ID: {character_id} - Image Path: {image_path} - Does not exist")
```



Screenshot 9:

Creating a Function to Test Photo Path

- *Description:* The screenshot displays the definition of a function for testing photo paths, which outputs an appropriate message depending on the results.

Screenshot 10:

Outcome of the Photo Path Testing Function

- *Description:* The screenshot shows the result of the photo path testing function. The status code for all photo paths is 200.

The test has passed.

Testing Character Data Modification Using POST Requests

GOT_POST_Data_Modification.py > ...

```python
import requests

path = "https://thronesapi.com/api/v2/Characters"

data = {
    "id": "1",
    "firstName": "New_first_name",
    "lastName": "New_last_name",
    "fullName": "New_full_name",
    "title": "New_title",
    "family": "New_family",
    "image": "New_image",
    "imageUrl": "https://thronesapi.com/assets/images/new_image_path.jpg"
}

# sending a POST request with data
post_response = requests.post(path, json=data)  # Setting the request body in JSON format
print("\nStatus code of the POST request is:", post_response.status_code)

# checking the status code for the POST request
if post_response.status_code == 200:
    print("Data added successfully via a POST request.")
else:
    print("POST request encountered an error with status code:", post_response.status_code)

# sending an empty POST request
empty_post_response = requests.post(path)
print("Status code of the empty POST request is:", empty_post_response.status_code)

# checking the status code for the empty POST request
if empty_post_response.status_code == 200:
    print("Empty POST request succeeded.")
else:
    print("Empty POST request encountered an error with status code:", empty_post_response.status_code)

# sending a GET request to retrieve data about the character with ID 1
get_response = requests.get(f"{path}/1")

# checking the status code for the GET request
if get_response.status_code == 200:
    print("Response to the GET request is:", get_response.json())

    # checking if the character data has been updated
    if get_response.json() == data:
        print("Data has been successfully updated.\n")
    else:
        print("Data has not been updated.\n")
else:
    print("GET request did not succeed.\n")
```

Screenshot 11:

Creating a Function for Modifying Character Data with POST Requests

- *Description:* The screenshot displays the definition of a function for modifying character data using POST requests. The function also tests the status code of GET and POST requests without a body, as well as the result of any data changes.

The output is:

```
The status code of the POST request is: 200
Data was added successfully via a POST request.
The status code of the empty POST request is: 415
Empty POST request encountered an error with status code: 415
Response to the GET request is: {'id': 1, 'firstName': 'Samwell', 'lastName': 'Tarly', 'fullName': 'Samwell Tarly', 'title': 'Maester', 'family': 'House Tarly', 'image': 'sam.jpg', 'imageUrl': 'https://thronesapi.com/assets/images/sam.jpg'}
Data has not been updated.
```

The test did not pass.