Report on the Analysis of fxhash Generative Artworks

Author: Iqra Kalhoro
Date: 21nd Sept 2024.

Table of Content:

1. Introduction

The growing field of generative art is powered by code rather than traditional images. The fxhash is a popular platform for generative artists to mint and distribute their work, where the unique visual outputs are generated through algorithms running in the browser. This report presents a detailed explanation of the script designed to collect, analyze, and save information from generative artworks on fxhash using Python and several web technologies.

The primary objective of this script is to automate the process of fetching artwork details from fxhash, analyze the libraries used (especially p5.js and its versions), and retrieve artwork data stored on IPFS. This is essential for understanding the technological dependencies of generative art and identifying issues related to artwork availability, outdated libraries, and IPFS file failures.

1.1. Scope and Objective

1. Automate the collection of artwork data from fxhash, using the platform's API and web scraping techniques.
2. Extract and analyze the creative coding libraries used by these artworks, with a focus on the p5.js library.
3. Fetch data from IPFS links, convert them into HTTP-accessible formats, and analyze broken links and missing artworks.
4. Provide a dataset with details of each artwork, including links, p5.js versions, IPFS URIs, and any identified libraries.

2. Overview of Technologies and Tools Used

2.1. Python Libraries and Their Roles

1. Selenium: Used for automating browser interactions, such as loading artwork pages in Chrome and waiting for elements to load. Selenium provides robust tools for web automation and scraping dynamic content.
2. BeautifulSoup: Used for parsing HTML responses and extracting content, such as artwork descriptions, links, and metadata from the fxhash platform.
3. Requests: A simple and efficient HTTP library used for making API calls and downloading resources like IPFS-hosted artwork files.
4. pandas: Used to manage data in tabular form and save it as CSV files. This enables exporting analyzed data for further use.
5. re: The Python Regular Expressions module is used to search for specific patterns in the code, especially for identifying p5.js versions and other JavaScript libraries.

2.2. WebDriver Manager and Chrome Options

The **Chrome WebDriver** automates Chrome browser interaction. The script initializes a Chrome session, navigates to fxhash artwork pages, and retrieves data. The **WebDriverManager** package ensures that the correct version of the ChromeDriver is installed and used.

2.3. IPFS (InterPlanetary File System)

fxhash uses IPFS to store generative code and other metadata in a decentralized manner. To interact with this system, the script converts IPFS links into HTTP links that are accessible through gateways like `gateway.ipfs.io` or `gateway.fxhash2.xyz`.

—

3. fxhash API and Web Scraping Strategy

3.1. fxhash Public API

The fxhash API provides metadata for artworks, including artwork descriptions, IPFS links, thumbnail URIs, display URIs, and more. The script uses the following API endpoint:

https://api.fxhash.xyz/v1/tokens/{artwork_id}

This endpoint returns artwork data in JSON format, which the script parses to extract various fields.

3.2. Web Scraping for Artwork Pages

When the API does not provide sufficient information or is unavailable, the script falls back on web scraping. Using Selenium and BeautifulSoup, the script navigates to artwork pages, extracts IPFS links, and parses the content for relevant metadata and libraries. The scraping logic captures:

1. Artwork descriptions (HTML element: `div` with class `GenerativeDisplay_description`).
2. IPFS links found in `a` tags or embedded scripts in the page.

4. Implementation Details

The core logic of the script revolves around several major functions:

4.1. `ipfs_to_http` – Converting IPFS Links

IPFS links (e.g., `ipfs://Qm...`) are not directly accessible through a browser. The `ipfs_to_http` function converts these links into HTTP-accessible formats via IPFS gateways. This is crucial for retrieving artwork files from decentralized storage.

Example:

```
def ipfs_to_http(ipfs_link):
    if ipfs_link.startswith("ipfs://"):
        http_link = f"https://gateway.ipfs.io/ipfs/{ipfs_link[7:]}"
        fxhash_link = f"https://gateway.fxhash2.xyz/ipfs/{ipfs_link[7:]}"
        return http_link, fxhash_link
    return ipfs_link, ipfs_link
```

4.2. `fetch_artwork_from_api` – Querying the fxhash API

This function queries the fxhash API for artwork data. If successful, the JSON response includes metadata such as the artwork's IPFS link, description, and URIs.

Example:

```
def fetch_artwork_from_api(artwork_id):
    api_url = f"https://api.fxhash.xyz/v1/tokens/{artwork_id}"
    try:
        response = requests.get(api_url, timeout=5)
        if response.status_code == 200:
            return response.json()
        return None
    except requests.exceptions.RequestException:
        return None
```

4.3. `analyze_artwork` – Fetching and Analyzing Artwork Metadata

This is the primary function that coordinates the retrieval and analysis of artwork data. It first attempts to fetch data via the fxhash API. If the API does not return sufficient data, the function switches to web scraping using Selenium and BeautifulSoup.

This function also:

1. Extracts creative coding libraries like p5.js and other JavaScript libraries from the code content.
2. Identifies and converts various URIs (artifact, thumbnail, display) from IPFS format to HTTP format.

Example:

```
def analyze_artwork(url, artwork_id):
    api_data = fetch_artwork_from_api(artwork_id)
    if api_data and 'token' in api_data:
        token = api_data['token']
        ipfs_link = token.get('ipfs', '-')
        code_content = fetch_ipfs_code(ipfs_link)
        p5_version_summary, other_libraries = extract_libraries(code_content)
        return description_text, p5_version_summary, other_libraries, ipfs_link
```

4.4. `fetch_ipfs_code` – Downloading IPFS Content

This function retrieves the generative code stored in IPFS links. It sends an HTTP request to the converted IPFS HTTP link and returns the code content.

Example:

```
def fetch_ipfs_code(ipfs_link):
    try:
        code_response = requests.get(ipfs_link)
        return code_response.text if code_response.status_code == 200 else None
    except Exception:
        return None
```

5. Analysis of Generative Artworks

5.1. Breakdown of p5.js Versions

p5.js is a popular creative coding library that many artists use to generate interactive visuals. The script identifies the specific versions of p5.js used in the generative artworks. The results from analyzing multiple artworks show that:

p5.js Version 1.4.0 was the most common version found.
Older versions like p5.js 0.8.0 were also detected in a few artworks.
In some cases, no p5.js library was used, and other JavaScript libraries (e.g., three.js) were detected.

The reliance on specific library versions makes some artworks susceptible to becoming broken as browser APIs evolve.

5.2. Broken Artworks

1. IPFS Link Failures: Some artworks had broken IPFS links, resulting in an inability to retrieve the generative code.
2. Rendering Issues: Certain artworks using outdated versions of p5.js or other libraries no longer rendered correctly in modern browsers.
3. Missing Artifacts: A few artworks had missing or incomplete metadata, making it difficult to fully analyze their content.

6. Data Export

The script saves all extracted data into a CSV file, allowing further analysis. Each row in the CSV contains details for a specific artwork, including the following fields:

1. Generative Art Link: The URL to the artwork on fxhash.
2. Link Status: Whether the artwork loaded successfully.
3. Generative Library: A description of the creative coding libraries used in the artwork.
4. IPFS Code Link: The link to the IPFS-hosted code.
5. p5.js Version: The specific version of p5.js detected.
6. Other Libraries: Any other JavaScript libraries used in the artwork.
7. Artifact URI (HTTP): The HTTP-accessible link for the artwork artifact.
8. Display URI (HTTP): The HTTP link to the artwork display file.
9. Thumbnail URI (HTTP): The HTTP link to the artwork thumbnail.

7. Challenges and Limitations

7.1. IPFS Availability

One of the significant challenges is the availability of IPFS nodes. Since IPFS is a decentralized system, some nodes may not be available at the time of retrieval, causing temporary or permanent failures in fetching artwork code.

7.2. Dynamic Web Pages

The fxhash platform heavily relies on dynamic web content. The use of JavaScript to load key elements of the page makes web scraping more challenging. To address this, the script uses Selenium to wait for content to load.

7.3. Obfuscated Code

Some artworks contain minified or obfuscated JavaScript code,
making it difficult to parse and extract library versions or analyze their structure.

8. Conclusion

This project automates the process of collecting and analyzing generative artworks on fxhash, providing valuable insights into the libraries and technologies used. It demonstrates how decentralized storage systems like IPFS interact with generative art platforms and exposes the potential pitfalls of relying on external libraries for long-term artwork preservation.

Further enhancements could focus on improving error handling for IPFS link failures and expanding the library detection capabilities to include a wider array of creative coding frameworks.

References

- fxhash API Documentation:
https://github.com/fxhash/fxhash-api
- IPFS Documentation: https://docs.ipfs.io
- Selenium Documentation:
https://www.selenium.dev/documentation/