Rarible

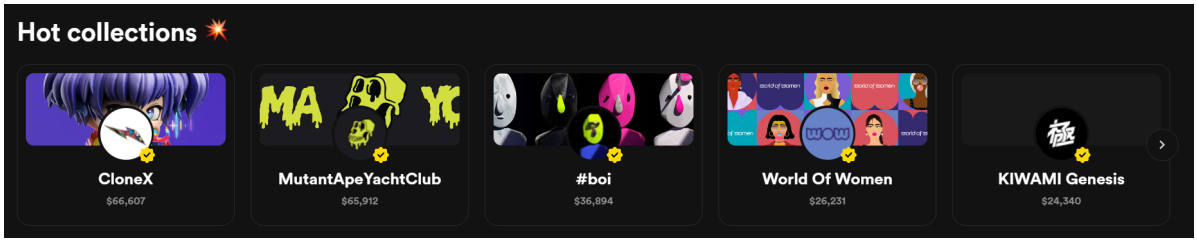
Aa Title	Tags
Medium	Web
Type of scraper	DOM Parsing, Text Pattern Matching
Type of crawling	Focused
Depth and Scraping Strategy	DFS, Shallow
Difficulty	Beginners
Technology	Javascript, Puppeteer

Rarible is a marketplace allowing digital artists and creators to issue and sell custom crypto assets that represent ownership in their digital work called NFTs. Our target is to download these artworks.

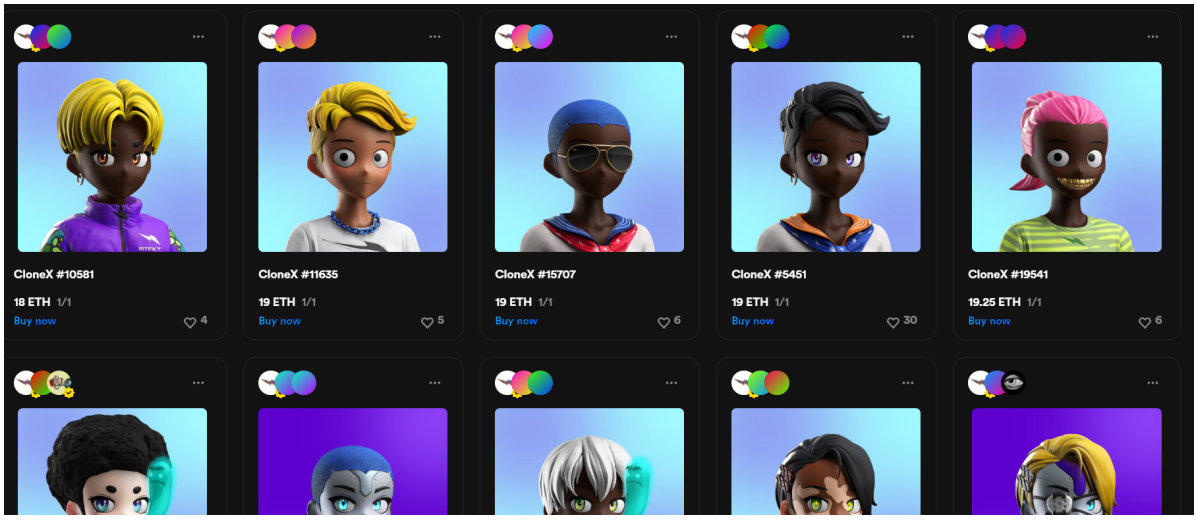
We will achieve data acquisition by scraping the Rarible website. The following is a detailed process of how we acquire the raw material to implement the task.

Process

1. Visit the Rarible website.
2. You will come across a “Hot Collections” section. Navigate to any collection of your choice.



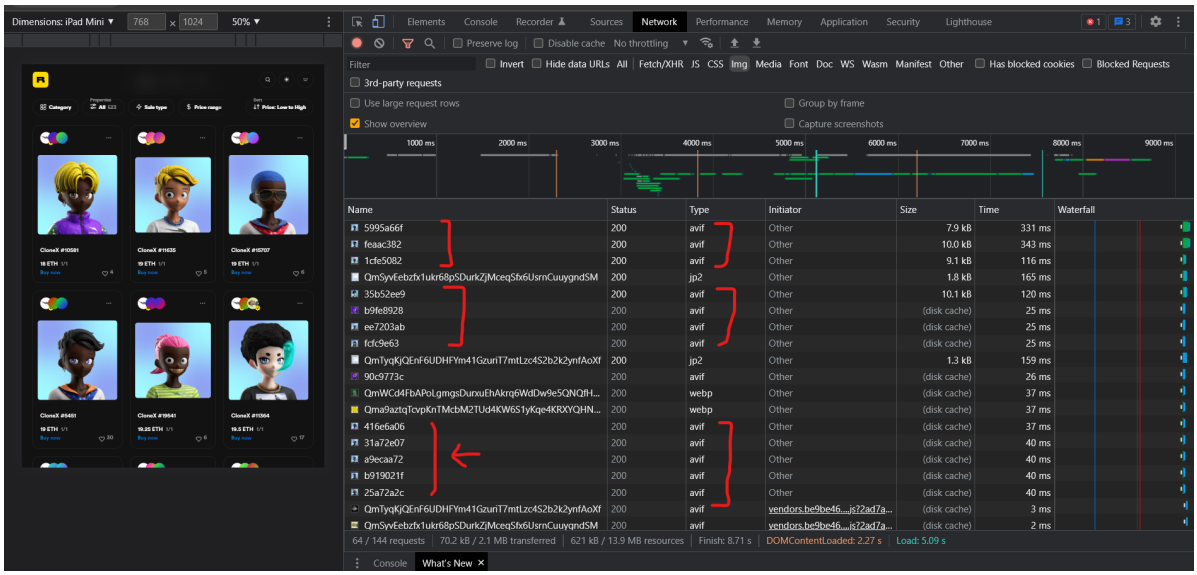
3. We have chosen CloneX. Now, as you scroll down, you'll see the entire collection of on-chain collectibles. We will download all these images using web scraping.



4. We will use the puppeteer driver to load the webpage source, extract links from all "<img>" tags and download all images. But there is a catch in this process. We do not want to download all the images on the page. Images in the header, logos, and user images should all be skipped and need not be downloaded.

   Thus, we will have to find some patterns in the image tags before downloading the images.

5. Load the Inspect Elements in the browser and go to the "Network" tab.
6. Reload the page. On reloading, all the images will reload and appear in the Network tab.



7. It is clear that images have appeared in the Network tab. These images are in AVIF format. Our task is to find the similarity factors between these images.

   Open the top five images in new tabs.

8. Yes, all these images have the same prefix but the suffix changes from: ”https://img.rarible.com/prod/image/upload/” + some suffix

   The suffix that defines the collection images is “t_image_preview”.

   eg:-

   - https://img.rarible.com/prod/image/upload/t_image_preview/prod-itemImages/0x49cf6f5d44e70224e2e23fdcdd2c053f30ada28b:4675/1cfe5082
   - https://img.rarible.com/prod/image/upload/t_image_preview/prod-itemImages/0x49cf6f5d44e70224e2e23fdcdd2c053f30ada28b:16/feaac382
   - https://img.rarible.com/prod/image/upload/t_image_preview/prod-itemImages/0x49cf6f5d44e70224e2e23fdcdd2c053f30ada28b:18633/5995a66f

9. Using regular expressions, we shall only download images from the collection.

Before moving to the implementation, we shall understand the characteristics of our scraping approach for Rarible.

- Rarible is built using React JS which runs on a virtual DOM technology. Here, we are scraping data that loads in real-time. Thus, the data scraper classifies into the DOM Parsing technique. Also, we are further setting a filter on data using text patterns in the link of the images. This sub-classifies the scraper into a Text pattern matcher.
- Our objective is to download only collectibles images. So, we have set filters on the crawler. Thus, we classify it as a focused crawling.
- As scraping is a single layer scraping, we classify it as shallow scraping in DFS fashion.

Implementation

1. For this implementation, we shall write a JavaScript script to read the webpage source. We'll initiate by instantiating the web driver using Puppeteer.
2. As Rarible is built over React JS, we can't directly start scraping as we would miss out first few data blocks. This is because React JS is based on virtual DOM technology. Thus, we will have to wait until the Virtual DOM loads.

   ```
   await page.waitForSelector(".ReactVirtualized__Grid");
   ```

3. As you scroll down the page, you will find discover a "Load more" button. We will either have to press the load more button manually or automate it. Automation is a better option to increase scraping speed. We will enable the web driver to press the "Load more" button every time the scraper encounters it. This would load the data on the next page of the flat list.

   ```
   const elements = [...document.querySelectorAll("button")];
   const targetElement = elements.find((e) => e.innerText.includes("Load more"));
   targetElement && targetElement.click();
   ```

4. We will filter and download the image links with the word “t_image_preview” present in them. (As explained earlier.)

   `if (imageUrl.includes("t_image_preview")) {...};`

5. We have to enable auto-scroll so that the data keeps loading.

   `await autoScroll(page);`

The following are a few snapshots of the program outcome:

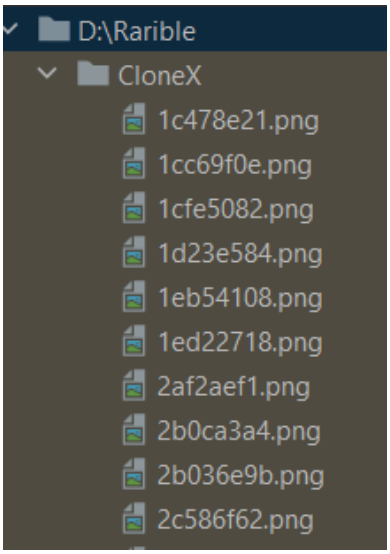
npm process starts scraping and saving the images to the CloneX folder.

```
PS D:\Rarible> npm start https://rarible.com/collection/0x49cf6f5d44e70224e2e23fdcdd2c053f30ada28b/

> nftgrabber@1.0.0 start D:\Rarible
> node index.js "https://rarible.com/collection/0x49cf6f5d44e70224e2e23fdcdd2c053f30ada28b/"

Loading...
CloneX #1 saved to CloneX/99a5d299.png
CloneX #2 saved to CloneX/1cfe5082.png
CloneX #3 saved to CloneX/90c9773c.png
CloneX #4 saved to CloneX/b9fe8928.png
CloneX #5 saved to CloneX/fcfc9e63.png
CloneX #6 saved to CloneX/774e70a9.png
CloneX #7 saved to CloneX/5995a66f.png
CloneX #8 saved to CloneX/ae81250e.png
CloneX #9 saved to CloneX/a334efae.png
CloneX #10 saved to CloneX/6839128f.png
```

Downloaded Images:



The entire dataset can be accessed here.

Thank you!