

JavaScript web scraper is the preferred choice due to its advantages working on dynamic sites over python and the familiarity of the language compared to python.

<https://github.com/BBarrettGreening/web-scraper>

Example scraper designed to scrape the name, dates and locations of shows from the Shakespeare Globe theatre website.

Postman initially used to send POST requests to the scrape endpoint with the website URL given in the request body

The screenshot shows the Postman application interface. A POST request is being made to `http://localhost:3000/scrape`. The request body contains the following JSON:

```
1 {
2   "url": "https://www.shakespearesglobe.com/whats-on/"
3 }
```

The response status is `200 OK`, and the response body is:

```
1 {
2   "events_saved": 39,
3   "message": "Events scraped successfully",
4   "filename": "2024-11-12-19-55.json"
5 }
```

The Postman interface also displays the request headers and the raw log of the request.

The application successfully scraped all 30 productions on the website and saved them in a JSON file when running on the localhost via node.js – Need list of attributes Stagely wants from all sites.

```
{  
    "title": "All's Well That Ends Well",  
    "date": "12 November - 4 December",  
    "link": "https://www.shakespearesglobe.comhttps://www.shakespearesglobe.com/whats-on/allss-well-that-ends-well/"  
},  
{  
    "title": "Chekhov's Short Stories",  
    "date": "1 December",  
    "location": "Sam Wanamaker Playhouse",  
    "link": "https://www.shakespearesglobe.comhttps://www.shakespearesglobe.com/whats-on/chekhovs-short-stories/"  
},  
{  
    "title": "A Night In Sign",  
    "date": "8 December",  
    "location": "Sam Wanamaker Playhouse",  
    "link": "https://www.shakespearesglobe.comhttps://www.shakespearesglobe.com/whats-on/a-night-in-sign/"  
},  
{  
    "title": "The Globe Talks: Simon Armitage in Conversation",  
    "date": "15 December",  
    "location": "Sam Wanamaker Playhouse",  
    "link": "https://www.shakespearesglobe.comhttps://www.shakespearesglobe.com/whats-on/simon-armitage-in-conversation/"  
},  
{  
    "title": "Tim Key: Chrimbo Bimbo",  
    "date": "22 - 23 December",  
    "location": "Sam Wanamaker Playhouse",  
    "link": "https://www.shakespearesglobe.comhttps://www.shakespearesglobe.com/whats-on/tim-key-chrimbo-bimbo/"  
},  
{  
    "title": "Cymbeline",  
    "date": "10 January - 20 April",  
    "location": "Sam Wanamaker Playhouse",  
    "link": "https://www.shakespearesglobe.comhttps://www.shakespearesglobe.com/whats-on/cymbeline/"  
},  
}
```

Currently could be possible to do multiple sites by making the fields it is parsing from dynamic and allow any url in the request then the request body would be formatted like this.

- Would be very time consuming to define a layout for each site but possible

```
55
56 [
57   "sites": [
58     {
59       "url": "https://www.shakespearesglobe.com/some-page",
60       "selectors": {
61         "title": ".c-event-card__title",
62         "date": ".c-event-card__date",
63         "location": ".c-event-card__venue",
64         "link": ".c-event-card__cover-link"
65       }
66     },
67     {
68       "url": "https://www.anotherwebsite.com/events",
69       "selectors": {
70         "title": ".event-title",
71         "date": ".event-date",
72         "location": ".event-location",
73         "link": ".event-link"
74       }
75     }
76   ]
77 ]
```

Ideally another service will scan the site HTML and return the attributes we're searching for to be passed to the scraper body. (Dynamically handle site layout changes)

The saved JSON outputs can be handled by another service which would sort the shows into a database (SQL database to sort in Stagely's preferred format)

The database sorted by shows could then be accessed and pushed to the content management system by another service (Another API which takes information from the database and autofill sections for the user)

To prevent having to use a third-party website to access the API endpoints swagger can also be implemented which will generate a UI for the web scraper actions.

The screenshot shows a Swagger UI interface for a 'Scraping API'. The top navigation bar includes links for 'Dashboard', 'Home Page - Univer...', and 'Full stack open'. The main title is 'Show Scraper API' with version '1.0.0' and 'OAS 3.0'. Below the title, it says 'API for scraping product and event data'. A 'Servers' dropdown is set to 'http://localhost:3000'. The main content area is titled 'default' and contains a 'POST /scrape' operation. The description states: 'Scrapes shows, location and dates from the specified URL.' Under 'Parameters', it says 'No parameters'. Under 'Request body' (marked as required), there is a schema definition:

```
{ "url": "string" }
```

. A 'Try it out' button is available. The 'Responses' section shows a single entry for code 200: 'A JSON object with the number of events saved.' There are no links listed under the responses. The bottom of the screen shows a Windows taskbar with various pinned icons and a weather widget indicating '12°C Mostly sunny'.

```
8 // Swagger setup
9 const swaggerUi = require('swagger-ui-express');
10 const swaggerJSDoc = require('swagger-jsdoc');
11
12 const swaggerOptions = {
13   swaggerDefinition: {
14     openapi: '3.0.0',
15     info: {
16       title: 'Show Scraper API',
17       version: '1.0.0',
18       description: 'API for scraping product and event data',
19     },
20     servers: [
21       {
22         url: `http://localhost:${port}`,
23       },
24     ],
25   },
26   apis: ['./src/server.js', './src/scraping.js'],
27 };
28
29 /**
30 * @swagger
31 * /scrape:
32 *   post:
33 *     summary: Scrapes shows, location and dates from the specified URL.
34 *     requestBody:
35 *       required: true
36 *       content:
37 *         application/json:
38 *           schema:
39 *             type: object
40 *             properties:
41 *               url:
42 *                 type: string
43 *                 description: The URL to scrape.
44 *     responses:
45 *       200:
46 *         description: A JSON object with the number of events saved.
47 *         content:
48 *           application/json:
49 *             schema:
50 *               type: object
51 *               properties:
52 *                 events_saved:
53 *                   type: integer
54 *                   description: The number of events scraped and saved.
55 *                 message:
56 *                   type: string
57 *                   description: Success message.
58 *                 filename:
59 *                   type: string
60 *                   description: Name of the file where events are saved.
61 *       400:
62 *         description: Invalid URL provided.
63 *       500:
64 *         description: Error scraping events.
65 */
```