Features

Crawl

Firecrawl can recursively search through a urls subdomains, and gather the content

Firecrawl thoroughly crawls websites, ensuring comprehensive data extraction while bypassing any web blocker mechanisms. Here's how it works:

1. **URL Analysis:** Begins with a specified URL, identifying links by looking at the sitemap and then crawling the website. If no sitemap is found, it will crawl the website following the links.
2. **Recursive Traversal:** Recursively follows each link to uncover all subpages.
3. **Content Scraping:** Gathers content from every visited page while handling any complexities like JavaScript rendering or rate limits.
4. **Result Compilation:** Converts collected data into clean markdown or structured output, perfect for LLM processing or any other task.

This method guarantees an exhaustive crawl and data collection from any starting URL.

Crawling

/crawl endpoint

Used to crawl a URL and all accessible subpages. This submits a crawl job and returns a job ID to check the status of the crawl.

🔥 Firecrawl

Features › **Crawl**

Installation

Python | Node | Go | Rust

```
pip install firecrawl-py
```

Usage

Python | Node | Go | Rust | cURL

```python
from firecrawl import FirecrawlApp

app = FirecrawlApp(api_key="fc-YOUR_API_KEY")

# Crawl a website:
crawl_status = app.crawl_url(
  'https://firecrawl.dev',
  params={
    'limit': 100,
    'scrapeOptions': {'formats': ['markdown', 'html']}
  },
  poll_interval=30
)
print(crawl_status)
```

Response

If you're using cURL or `async crawl` functions on SDKs, this will return an `ID` where you can use to check the status of the crawl.

Firecrawl

Features › **Crawl**

```
}
```

Check Crawl Job

Used to check the status of a crawl job and get its result.

> This endpoint only works for crawls that are in progress or crawls that have completed recently.

Python | Node | Go | Rust | cURL

```python
crawl_status = app.check_crawl_status("<crawl_id>")
print(crawl_status)
```

Response Handling

The response varies based on the crawl's status.

For not completed or large responses exceeding 10MB, a `next` URL parameter is provided. You must request this URL to retrieve the next 10MB of data. If the `next` parameter is absent, it indicates the end of the crawl data.

The skip parameter sets the maximum number of results returned for each chunk of results returned.

> The skip and next parameter are only relavent when hitting the api directly. If you're using the SDK, we handle this for you and will return all the results at once.

Scraping | Completed

Firecrawl

Features › **Crawl**


```
  "creditsUsed": 10,
  "expiresAt": "2024-00-00T00:00:00.000Z",
  "next": "https://api.firecrawl.dev/v1/crawl/123-456-789?skip=10",
  "data": [
    {
      "markdown": "[Firecrawl Docs home page![light logo](https://mintlify.s3-us-
      "html": "<!DOCTYPE html><html lang=\"en\" class=\"js-focus-visible lg:[--sc
      "metadata": {
        "title": "Build a 'Chat with website' using Groq Llama 3 | Firecrawl",
        "language": "en",
        "sourceURL": "https://docs.firecrawl.dev/learn/rag-llama3",
        "description": "Learn how to use Firecrawl, Groq Llama 3, and Langchain t
        "ogLocaleAlternate": [],
        "statusCode": 200
      }
    },
    ...
  ]
}
```


Crawl WebSocket

Firecrawl's WebSocket-based method, `Crawl URL and Watch`, enables real-time data extraction and monitoring. Start a crawl with a URL and customize it with options like page limits, allowed domains, and output formats, ideal for immediate data processing needs.

Python | Node

```
# inside an async function...
nest_asyncio.apply()

# Define event handlers
```

Firecrawl

Features › **Crawl**

```
def on_done(detail):
    print("DONE", detail['status'])

    # Function to start the crawl and watch process
async def start_crawl_and_watch():
    # Initiate the crawl job and get the watcher
    watcher = app.crawl_url_and_watch('firecrawl.dev', { 'excludePaths': ['blog/

    # Add event listeners
    watcher.add_event_listener("document", on_document)
    watcher.add_event_listener("error", on_error)
    watcher.add_event_listener("done", on_done)

    # Start the watcher
    await watcher.connect()

# Run the event loop
await start_crawl_and_watch()
```

Crawl Webhook

You can now pass a `webhook` parameter to the `/crawl` endpoint. This will send a POST request to the URL you specify when the crawl is started, updated and completed.

The webhook will now trigger for every page crawled and not just the whole result at the end.

cURL

```
curl -X POST https://api.firecrawl.dev/v1/crawl \
    -H 'Content-Type: application/json' \
    -H 'Authorization: Bearer YOUR_API_KEY' \
```

Webhook Events

There are now 4 types of events:

`crawl.started` - Triggered when the crawl is started.

`crawl.page` - Triggered for every page crawled.

`crawl.completed` - Triggered when the crawl is completed to let you know it's done (Beta)**

`crawl.failed` - Triggered when the crawl fails.

Webhook Response

`success` - If the webhook was successful in crawling the page correctly.

`type` - The type of event that occurred.

`id` - The ID of the crawl.

`data` - The data that was scraped (Array). This will only be non empty on `crawl.page` and will contain 1 item if the page was scraped successfully. The response is the same as the `/scrape` endpoint.

`error` - If the webhook failed, this will contain the error message.

**Beta consideration

There is a very tiny chance that the `crawl.completed` event may be triggered while the final `crawl.page` events are still being processed. We're working on a fix for this.

Firecrawl

Features › **Crawl**

Powered by Mintlify