FireCrawl

1.) It takes a URL, crawls it, and converts it into clean markdown or structured data. It crawls all accessible subpages and gives a clean data for each. No sitemap required.

2.) Handles dynamic content: Dynamic websites, JS-rendered sites, PDFs, images

3.) Features:

a. **Scrape:** Scrapes a URL and return its content
b. **Crawl:** Scrapes all the URLs of a web page and returns its content
c. **Map:** Input a website and get all the website URLs – extremely fast
d. **Search:** Search the web and get full content from the results
e. **Extract:** Get structured data from single page, multiple pages, or entire websites with AI.

4.) Powerful Capabilities

a. LLM-ready formats: markdown, structured data, screenshot, HTML, links, metadata
b. Actions: click, scroll, input, wait and more before extracting data

5.) Document Parsing:

a. Document parsing in Firecrawl works automatically when you provide a URL that points to a supported document type. The system will detect the file type based on the URL extension or content-type header and process it accordingly.
b. Excel Spreadsheets (.xlsx, .xls): Each worksheet is converted to an HTML table. Preserves cell formatting and data types

c. Word Documents (.docx, .doc, .odt, .rtf): Extracts text content while preserving document structure. Maintains headings, paragraphs, lists, and tables. Preserves basic formatting and styling
d. PDF Documents (.pdf): Extracts text content with layout information. Preserves document structure including sections and paragraphs. Handles both text-based and scanned PDFs (with OCR support).

6.) **Proxy**:

a. Firecrawl supports three types of proxies:
    i. basic: Proxies for scraping most sites. Fast and usually works.
    ii. enhanced: Enhanced proxies for scraping complex sites while maintaining privacy. Slower, but more reliable on certain sites. Cost of enhanced proxy is 5 credits.
    iii. auto: Firecrawl will automatically retry scraping with enhanced proxies if the basic proxy fails.
b. Default proxy: auto.
c. Using Enhanced as a Retry Mechanism:
    i. A common pattern is to first try scraping with the default proxy settings, and then retry with enhanced mode if you encounter specific error status codes (401, 403, or 500) in the metadata.statusCode field of the response. These status codes can be indicative of the website blocking your request.

7.) **Scraping Formats**:

a. Markdown (markdown)
b. Summary (summary)
c. HTML (html)
d. Raw HTML (rawHtml) (with no modifications)
e. Screenshot (screenshot, with options like fullPage, quality, viewport)
f. Links (links)
g. JSON (Json) - structured output

h. Images (images) - extract all image URLs from the page
i. Branding (branding) - extract brand identity and design system

8.) **Branding Format**:

a. It extracts comprehensive brand identity information from a webpage, including colors, fonts, typography, spacing, UI components, and more.

9.) **Change tracking**:

a. Using the changeTracking format, you can monitor changes on a website
b. It returns:
    i. previousScrapeAt: The timestamp of the previous scrape that the current page is being compared against (null if no previous scrape)
    ii. changeStatus: The result of the comparison between the two page versions
        1. new: This page did not exist or was not discovered before (usually has a null previousScrapeAt)
        2. same: This page's content has not changed since the last scrape
        3. changed: This page's content has changed since the last scrape
        4. removed: This page was removed since the last scrape
    iii. visibility:
        1. visible: This page is visible, meaning that its URL was discovered through an organic route (through links on other visible pages or the sitemap)
        2. hidden: This page is not visible, meaning it is still available on the web, but no longer discoverable via the sitemap or crawling the site. We can only identify invisible links if they had been visible, and captured, during a previous crawl or scrape

c. Advanced Options: changeTracking can be configured by passing an object in the format array. Eg:

```
{    type: 'changeTracking',
    modes: ['git-diff', 'Json'], // Enable specific change tracking modes
}
```

d. The git-diff mode provides a traditional diff format like Git's output. It shows line-by-line changes with additions and deletions marked. The structured JSON representation of the diff includes:
    i. files: Array of changed files (in web context, typically just one)
    ii. chunks: Sections of changes within a file
    iii. changes: Individual line changes with type (add, delete, normal)

e. JSON mode:
    i. The Json mode provides a structured comparison of specific fields extracted from the content. This is useful for tracking changes in specific data points rather than the entire content.

f. Matching Previous Scrapes: Previous scrapes to compare against are currently matched on the source URL, the team ID, the markdown format, and the tag parameter.
    i. For an effective comparison, the input URL should be the same as the previous request for the same content.
    ii. Crawling the same URLs with different includePaths/ excludePaths/ includeTags/ excludeTags/ onlyMainContent will have inconsistencies when using changeTracking.
    iii. Compared pages will also be compared against previous scrapes that only have the markdown format without the changeTracking format.
    iv. Comparisons are scoped to your team. If you scrape a URL for the first time with your API key, its changeStatus will always be new, even if other Firecrawl users have scraped it before.

10.) **Caching**:

a. Avoid storing: Set storeInCache to false if you do not want Firecrawl to cache/store results for this request.
b. When to Use Caching: Documentation, articles, product pages, Bulk processing jobs, Development, and testing, Building knowledge bases
c. When caching should be skipped: Real-time data (stock prices, live scores, breaking news), Frequently updated content, Time-sensitive applications.

SCRAPE

1.) const doc = await app.scrape(URL, scrapeOptions);

2.) The above API function scrapes a single webpage and it returns an object containing clean data in the format specified by scrapeOptions

3.) scrapeOptions: It is an optional parameter containing

a. format: It is an array specifying the format of the resultant data.
    i. The array contains strings like “markdown” or object like:
        1. JSON: { type: "Json", prompt, schema }
        2. Screenshot: { type: "screenshot", fullPage?, quality?, viewport? }
        3. Change tracking: { type: "changeTracking", modes?, prompt?, schema?, tag? } (requires markdown)
    ii. Default: [‘markdown’]
b. onlyMainContent: Boolean value; False returns full page content and true returns only main page content. Default: true.
c. includeTags: An array of HTML tags/classes/ids to include in the scrape.
d. excludeTags: An array of HTML tags/classes/ids to exclude from the scrape.
e. waitFor: Milliseconds of extra wait time before scraping (use sparingly). This waiting time is in addition to Firecrawl’s smart wait feature. Default: 0
f. maxAge: If a cached version of the page is newer than maxAge, Firecrawl returns it instantly; otherwise it scrapes fresh and updates the cache. Set 0 to always fetch fresh. Default: 172800000 (2 days)
g. timeout: Max duration in milliseconds before aborting. Default: 30000 (30 seconds)
h. location: {country: <country>(default: “US”), languages: Array of languages (default: language of that country [“en”])}

i. storeInCache: It is a Boolean value, if true it stores the result in cache (speeds up repeat scrapes via maxAge) otherwise not.
j. PDF parsing: Control parsing behavior. To parse PDFs, set parsers: ["pdf"].
    i. Cost: PDF parsing costs 1 credit per PDF page. To skip PDF parsing and receive the file as base64 (1 credit flat), set parsers: [].
    ii. Limit pages: To limit PDF parsing to a specific number of pages, use parsers: [{"type": "pdf", "maxPages": 10}].
k. Actions:
    i. Perform various actions on a web page before scraping its content.
    ii. It is important to almost always use the wait action before/after executing other actions to give enough time for the page to load.
    iii. Supported actions:
        1. wait - Wait for page to load: { 'type': 'wait', 'milliseconds': 1000} or { type: "wait", selector: string } (wait for a specific element to be visible using wait with a selector parameter)
        2. click - Click an element: { 'type': 'click', 'selector': '#accept', all?: Boolean }
        3. write - Type text into a field: { 'type': 'write', 'text': 'FireCrawl' } (element must be focused first with click)
        4. press - Press a keyboard key: { 'type': 'press', 'key': 'Enter' }
        5. scroll: Scroll the page: { type: "scroll", direction: "up" | "down", selector?: string } (if a specific element is to be scrolled, provide that element using selector)
        6. screenshot - Capture screenshot: { type: "screenshot", fullPage?: Boolean, quality?: number, viewport?: { width: number, height: number } }
        7. scrape - Scrape sub-element: { type: "scrape" }
        8. executeJavascript - Run JS code: { type: "executeJavascript", script: string }

9. pdf - Generate PDF: { type: "pdf", format?: string, landscape?: Boolean, scale?: number }

4.) Synchronous Batch Scraping:

a. The synchronous method will return the results of the batch scrape job
b. const job = await app.batchScrape(array_of_URLS, params);
c. params: It is an optional parameter containing
    i. options: optional scrapeOptions
    ii. pollInterval: Seconds between internal status polls
    iii. timeout: Max seconds to wait for completion before erroring.
    iv. pagination: It is an optional parameter object containing:
        1. autoPaginate: Boolean value.
            a. If true: fetches all pages automatically merges into single data array
            b. If false: next page is stored in the next attribute.

5.) Asynchronous Batch Scraping:

i. The asynchronous method will return a job ID that you can use to check the status of the batch scrape. We can use an infinite loop and check at regular intervals when this job gets completed.
ii. const batchStart = await app.startBatchScrape(array_of_URLS, params);
iii. params: It is an optional parameter containing:
    1. options: optional scrapeOptions
    2. webhook: Webhook Config (gets notified on completed, failed, or per-page events.)
iv. Use getBatchScrapeStatus(id) to check manually.
v. Manual Pagination:
    1. const status = app.getBatchScrapeStatus(id);
    2. Next page is stored in the status.next

vi. We have to manually poll for the first page but for the subsequent pages there is no need of polling they are already fetched.

CRAWL

1.) const res = await app.crawl(baseUrl, crawlerOptions);

2.) The above API function scrapes a webpage and its subpages and it returns an object containing clean data in the format specified by crawlOptions

3.) Calling the crawl method will submit a crawl job, wait for it to finish, and return the complete results for the entire site.

4.) Crawl Status:
   a. The crawl() returns a job ID which can be used to check status.
   b. const status = await app.getCrawlStatus("<crawl-id>");
   c. returns either "scraping" or "completed" or "cancellation"

5.) Cancel Crawl: const ok = await app.cancelCrawl("<crawl-id>");

6.) If the content is larger than 10MB or if the crawl job is still running, the response may include a next parameter, a URL to the next page of results.

7.) crawlerOptions: It is an optional parameter containing all the scrapeOptions attributes and:
   a. includePaths: An array of regex patterned paths to include in the crawl
   b. excludePaths: An array of regex patterned paths not to be included in the crawl
   c. maxDiscoveryDepth: It determines the maximum discovery depth for finding new URLs

d. crawlEntireDomain: Boolean value determining whether to explore across siblings/parents to cover entire domain. Default: false
e. allowExternalLinks: Boolean value determining whether to Follow links to external domains. Default: false
f. allowSubdomains: Boolean value determining whether to Follow subdomains of the main domain. Default: false
g. limit: Max number of pages to crawl. Default: 10000
h. delay: delay in seconds between scrapes

8.) Crawl a website with Web Sockets:

a. crawlUrlAndWatch(url [,params]);
b. Event Types:
    i. “document”: Fires each time Firecrawl successfully crawls and processes a new page.
    ii. “error”: Fires when any crawl error occurs (network failure, page timeout, parsing issue).
    iii. “done”: Fires once when the entire crawl completes (hits limit, exhausts links, or times out).

```
const response = await app.startCrawl(baseUrl, { limit: 2 });
const watcher = app.watcher(response.id, {
      kind: "crawl",
      pollInterval: 2,
      timeout: 120,
   });
watcher.on("document", async (doc) => console.log(doc)};
watcher.on("error", async (err) => console.log(err));
watcher.on("done", (state) => console.log("DONE",
state.status));
await watcher.start();
```

SEARCH

1.) app.search(query, searchOptions)

2.) Perform web searches and optionally scrape the search results in one operation.
    a. Choose specific sources (web, news, images)
    b. Search the web with customizable parameters (location, etc.)

3.) searchOptions: Optional parameter conataining scrapeOptions and:
    a. limit: limit the number of searches
    b. location: string
    c. categories: filter search results by specific categories, It is an array of:
        i. “github”: Search within GitHub repositories, code, issues, and documentation
        ii. “research”: Search academic and research websites (arXiv, Nature, IEEE, PubMed, etc.)
        iii. “pdf”: Search for PDFs
    d. sources: array like [‘web’, ‘news’, ‘images’]
        i. web: standard web results (default)
        ii. news: news-focused results
        iii. images: image search results
    e. timeouts: time spent per request
    f. Time based Search: Use the tbs parameter to filter results by time:
        i. qdr:h - Past hour
        ii. qdr:d - Past 24 hours
        iii. qdr:w - Past week
        iv. qdr:m - Past month
        v. qdr:y - Past year
        vi. Custom date range format: tbs: "cdr: 1, cd_min: 12/1/2024, cd_max: 12/31/2024"

MAP

1.) app.map(URL, mapOptions);

2.) It takes the starting URL as a parameter and returns all the links on the website.

3.) mapOptions:

a. search: Filter Links containing search
b. limit: Maximum number of links to return
c. sitemap: control sitemap usage during mapping. Values: only, include or skip. Default: 'include'
d. includeSubdomains: Boolean value whether to include subdomains of the website or not. Default: false
e. Location and Language: Specify country and preferred languages to get relevant content based on your target location and language preferences.

EXTRACT

1.) app.extract({urls, prompt, schema, …otherExtractOptions})

2.) It extracts specific data. Data can be extracted with pre-defined schema or without schema.

3.) If extract is called without schema. The llm chooses the structure of the data.
   a. Urls: List of urls from where data will be extracted.
   b. prompt: Help guide extraction.
   c. schema: JSON Schema for the structured output.
   d. required: an array of required fields from the schema

4.) Calling the extract method will submit an extraction job, wait for it to finish, and return the complete results for the entire site.

Webhook

1.) Webhooks let you receive real-time notifications as your operations progress, instead of polling for status.

Crawl	started, page, completed
Batch Scrape	started, page, completed
Extract	started, completed, failed

2.) Add a webhook Object to the request, webhook: {url, metadata: {any_key} ,events}

Field	Type	Required	Description
url	String	Yes	URL Endpoint
headers	object	No	Custom headers to include
metadata	object	No	Custom data included in payloads
events	array	No	Event types to receive (default: all)

3.) Timeouts & Retries

a. Endpoint must respond with a 2xx status within 10 seconds.
b. If delivery fails (timeout, non-2xx, or network error), Firecrawl retries automatically:
c. Retry Delay after failure
   1st 1 minute
   2nd 5 minutes
   3rd 15 minutes
d. After 3 failed retries, the webhook is marked as failed and no further attempts are made.

4.) Eg:

```
curl -X POST https://api.firecrawl.dev/v2/crawl \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer YOUR_API_KEY' \
```

```
  -d '{
    "url": "https://docs.firecrawl.dev",
    "limit": 100,
    "webhook": {
      "url": "https://your-domain.com/webhook",
      "metadata": {
        "any_key": "any_value"
      },
      "events": ["started", "page", "completed"]
    }
  }'
```

*****