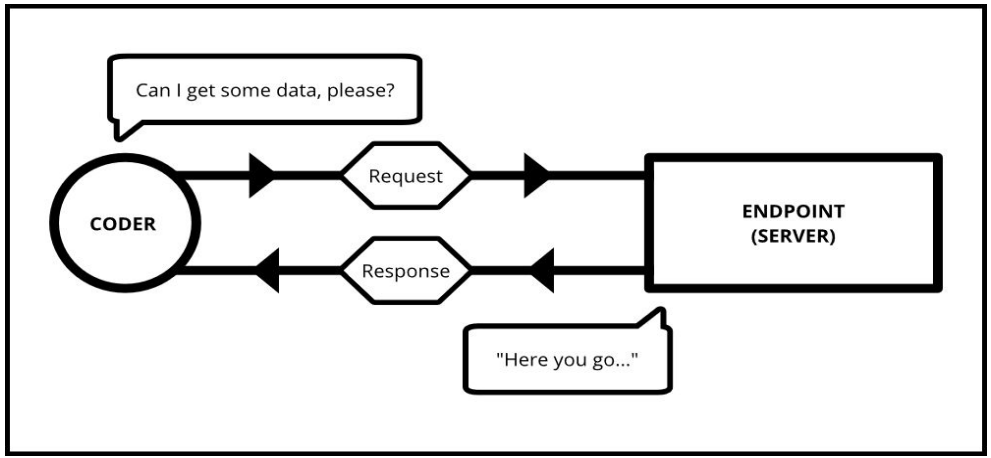
Fetching Data

“Stop trying to make fetch happen...”

What is Fetching Data?

Fetch is the retrieving of **data** by a software program, script, or hardware device. After being retrieved, the **data is** moved to an alternate location or represented on a device's screen.

Example: When you are searching for streaming content on a service such as Netflix, the main content will change, but most of the surrounding information, like the header, footer, navigation menu, etc., will stay the same. This saves time and processing power.



Fetching Resources

- In order to fetch data from resources, you use GET. This allows for you not to have to create new data that already exists in an outside resource.
- Example: We want specific data from a collection of articles. We can achieve this by requesting fetches from that collection.

```
GET /articles HTTP/1.1
Accept: application/vnd.api+json
// This is fetching the collection of articles from the JSON api
```

```
GET /articles/1 HTTP/1.1
Accept: application/vnd.api+json
// This is fetching the article with ID of 1 from the collection
```

```
GET /articles/1/author HTTP/1.1
Accept: application/vnd.api+json
// This is fetching the author from the article with ID of 1
```

Fetching Relationships

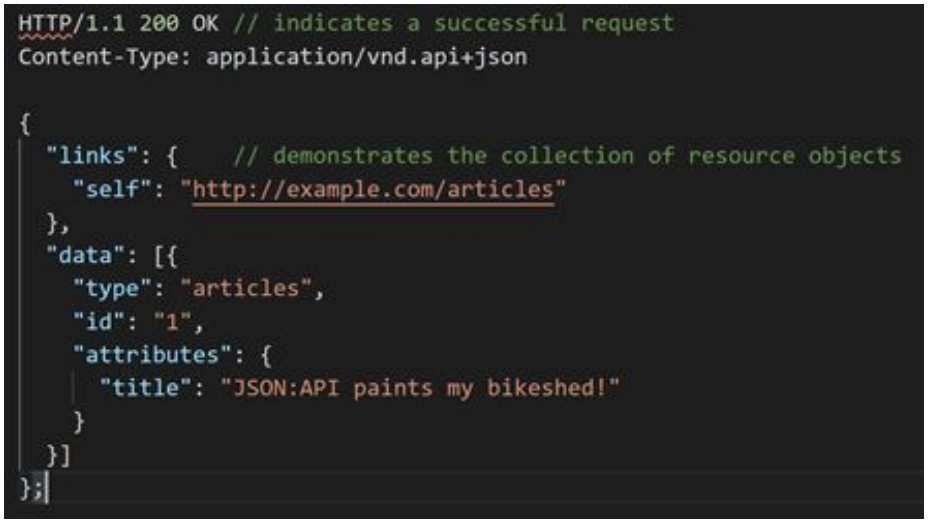
- A server MUST support fetching relationship data for every linked resource. Best practice would be to think of relationships as key value pairs. It could be displayed as “Key: Key Value” or “Author: Author of this article”.

- Example:

  You wouldn’t fetch the author of an article within a collection without also fetching the article itself first.

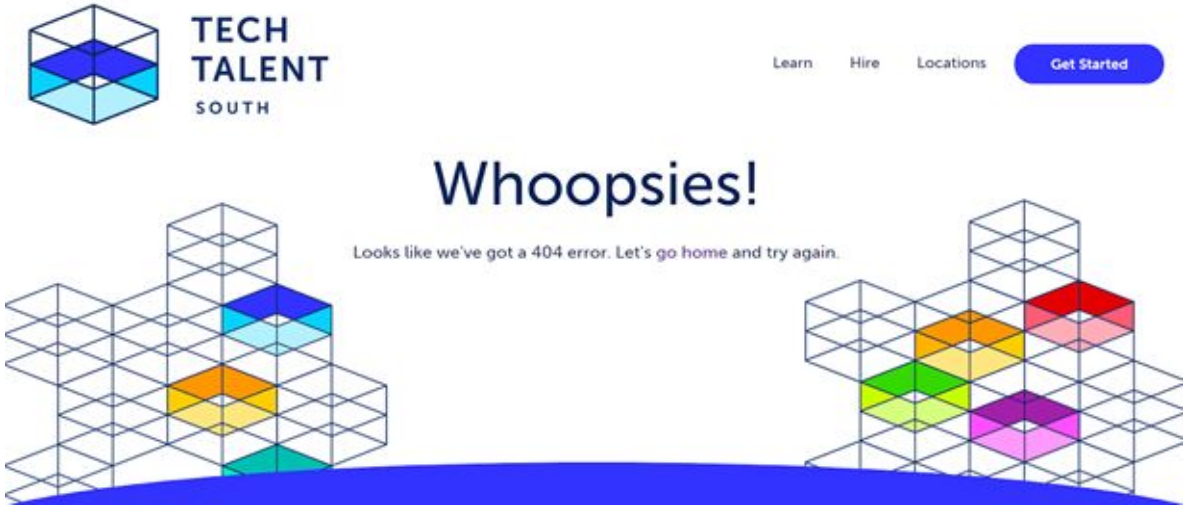
Responses (200 OK)

- A server will respond to all successful requests to fetch data from a resource. It will be indicated with a **200 OK** response.



Responses (404 Not Found)

- A server will respond with a 404 not found when processing a request to fetch a single resource that does exist.
- Responses similar to this one would be error details or HTTP status codes. A server MUST have these responses prepared and the client MUST be able to interpret.



Inclusion of Related Resources

- The endpoint also may support an **include** request parameter to allow the client to customize which related resources should be returned. If this can not be done, it with respond with a 400 error.

```
GET /articles/1?include=comments HTTP/1.1
Accept: application/vnd.api+json
// This is using an include request to fetch article with ID of 1
// as well as the comments associated to that article.
```

- In order to request resources related to other resources, a dot-separated path for each relationship name can be specified

```
GET /articles/1?include=comments.author HTTP/1.1
Accept: application/vnd.api+json
// This is also using the include request to fetch the comments
// but includes the author associated with those comments.
```

Sparse Fieldset

- A client may request that an endpoint return only specific fields in the response on a per-type basis by including a fields[TYPE] parameter that must be comma-separated.
- If a client requests specific set of fields for a response, an endpoint can’t include additional fields from the resource of that type in its response.

- Example: We would like to only include the title, body, and author of the articles in our return.

```
GET /articles?include=author&fields[articles]=title,body,author HTTP/1.1
// Here we want articles objects to have fields title, body and author only
// and people objects to have name field only.
```

Sorting

- A server may support a request to sort resources collections according to one or more criterias (“sort fields”) and allows you to order the results by any field, in ascending or descending order. If the server does not support the sorting in the query it must return the error code 400 Bad Report.

- Example: We want our fetched articles to be sorted by the date it was created.

```
GET /articles?sort=-created,title HTTP/1.1
Accept: application/vnd.api+json
// Here the return for this request with sort the articles from
// newest to oldest. If they were created on the same date, then
// they will be sorted alphabetically.
```

Pagination

- Pagination is the arrangement of numbers assigned to pages within a book. A server can limit the amount of data retrieved in a response to a subset (“page”) of the entire set available.

- Example: We want to fetch specifically page 3 of the article

```
GET /articles?page[number]=3&page[size]=1 HTTP/1.1
// Here it is requesting the return to include page 3 of the
// article and the size of the page is 1.
```

Filtering

- Basic filtering allows selecting resources by matching one or more members of the resource to values passed as query parameters.

- Example: We want to request all articles where Dale is the author.

```
GET /articles/articles?author=Dale HTTP/1.1
// This is fetching all articles within the collection that
// has the value of Dale for author.
```


YOU'RE LOOKING QUITE...
FETCHING