REST API Fundamentals - REST Implementation

CRUD

CRUD (Create, Read, Update, Delete) operations are fundamental to RESTful APIs as they correspond to the basic operations performed on resource

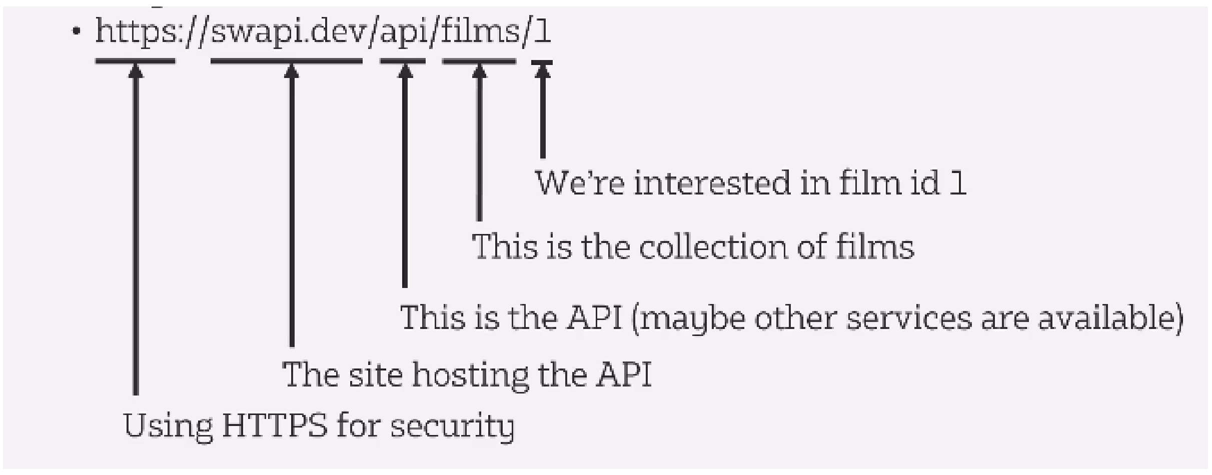
- Create
    - Insert a new record into the data store
- Read
    - Read an existing record or records from the data store
- Update
    - Modify an existing record in the data store
- Delete
    - Remove an existing record from the data store

- REST API is often just CRUD using HTTP/JSON

REST Endpoint

A REST endpoint is a specific URL (Uniform Resource Locator) that represents a resource, a collection of resources, or a service that can be accessed and manipulated by clients. REST endpoints are fundamental to the REST architectural style, as they define the entry points for interacting with the API's resources

URI (Uniform Resource Identifier): It's a generic term for identifying any resource, either online or offline,
URL (Uniform Resource Locator): A specific type of URI that provides both the identity and the location of an online resource, including details like the protocol, domain, path etc


• https://swapi.dev/api/films/1
We're interested in film id 1
This is the collection of films
This is the API (maybe other services are available)
The site hosting the API
Using HTTPS for security

Best practises for naming URLs and collections in RESTful API

- Use Plural Nouns for Collections
- Use Noun Phrases for Resource Names
- Avoid Verbs in URLs
- Use Hyphens or Underscores for Multi-Word Names
- Avoid Special Characters and Spaces
- Use Consistent Casing
- Version Your API
- Make URLs Hierarchical When Appropriate
- Use Singular Nouns for Specific Resources
- Consider SEO and User-Friendly URLs

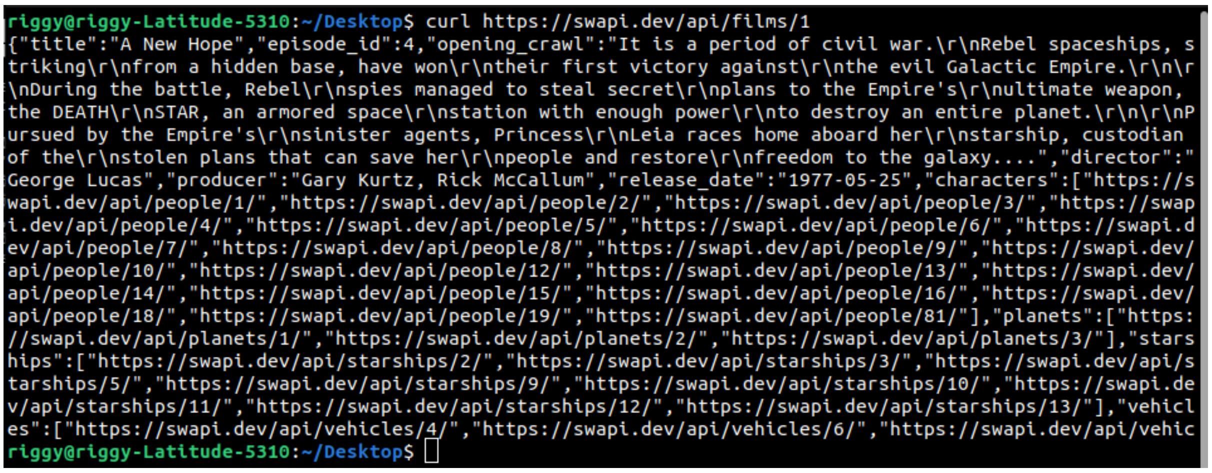
Nesting collections within collections

- Known as hierarchical resource structures
- Is a practice in RESTful API design where resources are organised in a hierarchical or nested manner
- Nesting collections within collections means organising resources in a parent-child relationship. For example, you can have a collection of "categories," and within each category, a collection of "products." This hierarchy is reflected in the URL structure and improves resource organisation and discoverability

Query String in URL with Name-Value Pairs

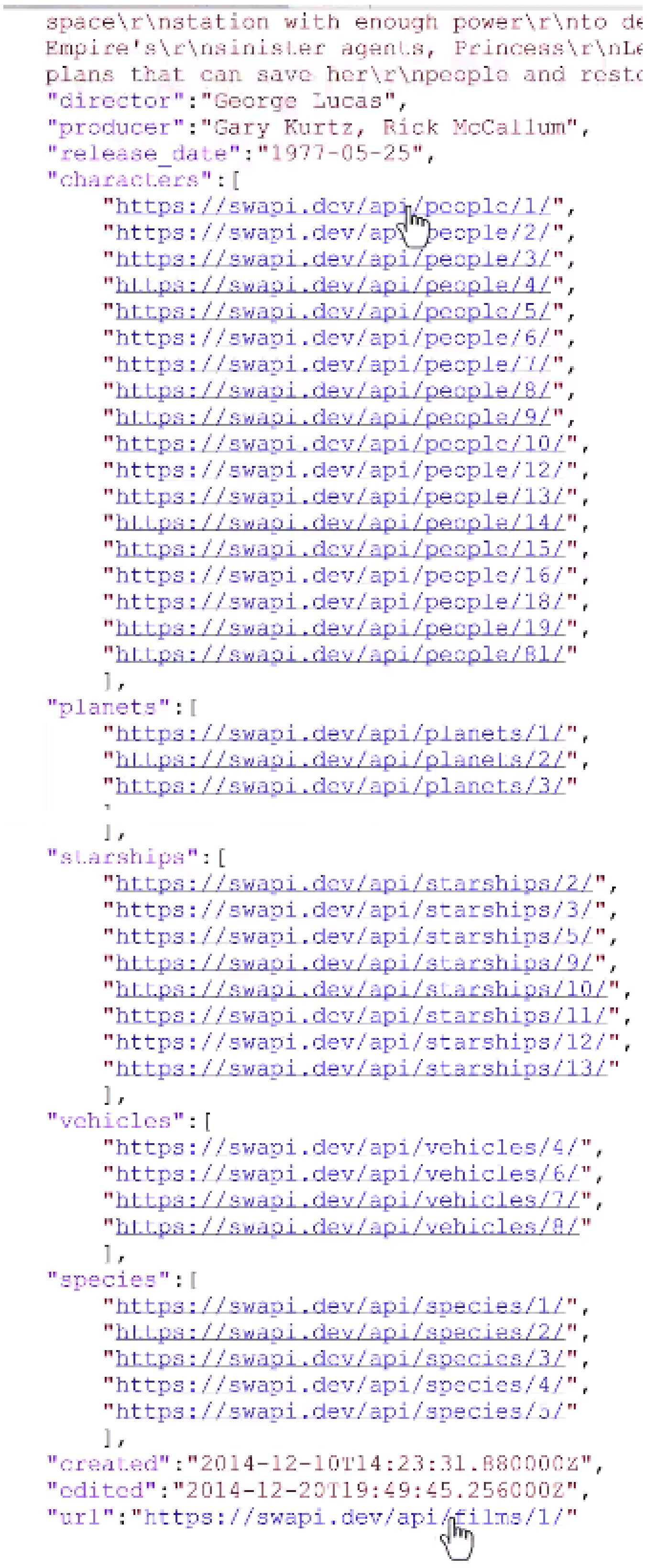
- Use a query parameter to filter results
- Added to the end of a URL after a question mark (?).
- Contains key-value pairs separated by ampersands (&).
- Used to provide data to a web server.
- Query strings are often used with HTTP GET requests to retrieve data from a server. The server can process the parameters and return results based on them.

<u>JSON document in Terminal and Notepad++ using Curl</u>



```
riggy@riggy-Latitude-5310:~/Desktop$ curl https://swapi.dev/api/films/1
{"title":"A New Hope","episode_id":4,"opening_crawl":"It is a period of civil war.\r\nRebel spaceships, s
triking\r\nfrom a hidden base, have won\r\ntheir first victory against\r\nthe evil Galactic Empire.\r\n\r
\nDuring the battle, Rebel\r\nspies managed to steal secret\r\nplans to the Empire's\r\nultimate weapon,
the DEATH\r\nSTAR, an armored space\r\nstation with enough power\r\nto destroy an entire planet.\r\n\r\nP
ursued by the Empire's\r\nsinister agents, Princess\r\nLeia races home aboard her\r\nstarship, custodian
of the\r\nstolen plans that can save her\r\npeople and restore\r\nfreedom to the galaxy....","director":"
George Lucas","producer":"Gary Kurtz, Rick McCallum","release_date":"1977-05-25","characters":["https://s
wapi.dev/api/people/1/","https://swapi.dev/api/people/2/","https://swapi.dev/api/people/3/","https://swap
i.dev/api/people/4/","https://swapi.dev/api/people/5/","https://swapi.dev/api/people/6/","https://swapi.d
ev/api/people/7/","https://swapi.dev/api/people/8/","https://swapi.dev/api/people/9/","https://swapi.dev/
api/people/10/","https://swapi.dev/api/people/12/","https://swapi.dev/api/people/13/","https://swapi.dev/
api/people/14/","https://swapi.dev/api/people/15/","https://swapi.dev/api/people/16/","https://swapi.dev/
api/people/18/","https://swapi.dev/api/people/19/","https://swapi.dev/api/people/81/"],"planets":["https:
//swapi.dev/api/planets/1/","https://swapi.dev/api/planets/2/","https://swapi.dev/api/planets/3/"],"stars
hips":["https://swapi.dev/api/starships/2/","https://swapi.dev/api/starships/3/","https://swapi.dev/api/s
tarships/5/","https://swapi.dev/api/starships/9/","https://swapi.dev/api/starships/10/","https://swapi.de
v/api/starships/11/","https://swapi.dev/api/starships/12/","https://swapi.dev/api/starships/13/"],"vehicl
es":["https://swapi.dev/api/vehicles/4/","https://swapi.dev/api/vehicles/6/","https://swapi.dev/api/vehic
riggy@riggy-Latitude-5310:~/Desktop$
```

- We have an example of a REST API that conforms with the requirements of REST
- Client and Server separation - client is Curl and server is swapi.dev
- Statelessness - no state information specific to user is stored
- Cacheable - data can be cached and is unchanged frequently
- Layered system - No information about how data is being stored on the database that is running on the server
- - HATEOAS - links allow us to gain more information of other elements in the database
- Key and values provided
- Characters provided as an array where each element is a string - contains the API endpoints for a person (character in the film) and is an example of Hypermedia as the Engine of Application State (HATEOAS)



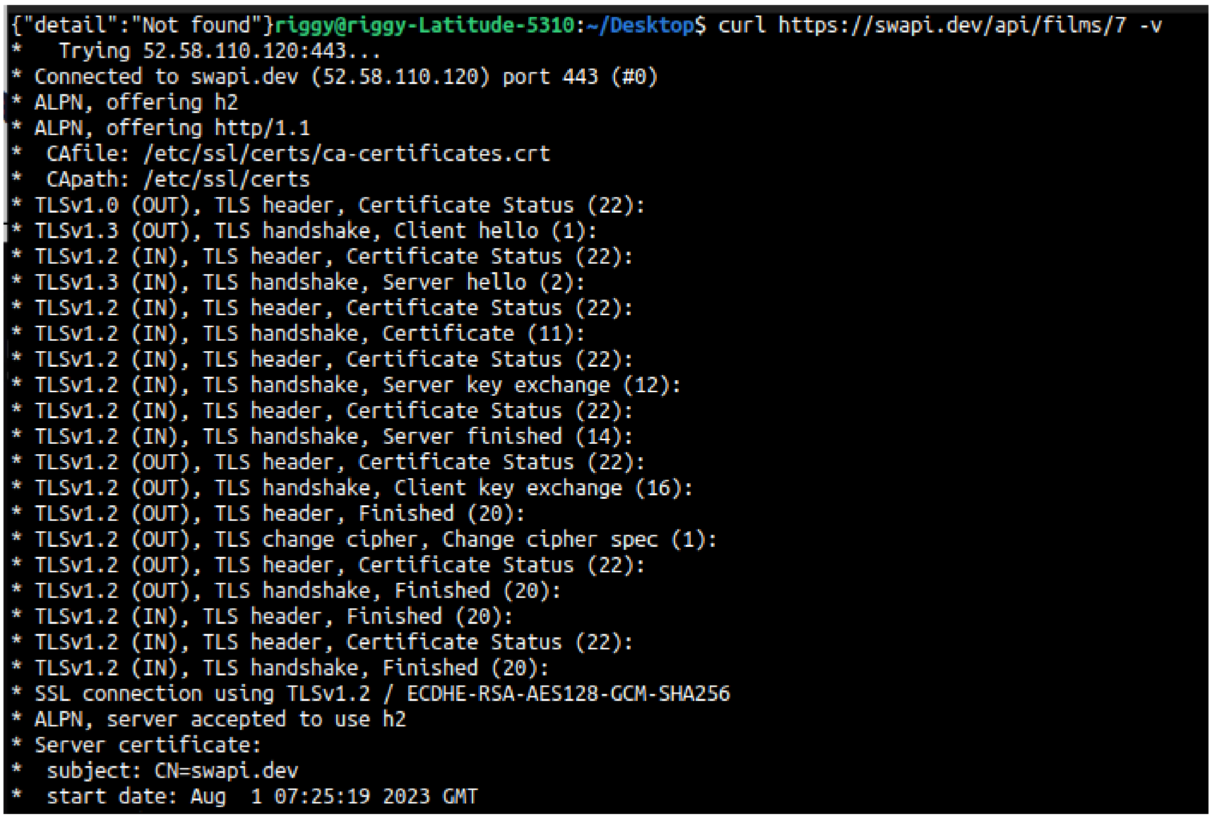
```
space\r\nstation with enough power\r\nto de
Empire's\r\nsinister agents, Princess\r\nLe
plans that can save her\r\npeople and resto
"director":"George Lucas",
"producer":"Gary Kurtz, Rick McCallum",
"release_date":"1977-05-25",
"characters":[
    "https://swapi.dev/api/people/1/",
    "https://swapi.dev/api/people/2/",
    "https://swapi.dev/api/people/3/",
    "https://swapi.dev/api/people/4/",
    "https://swapi.dev/api/people/5/",
    "https://swapi.dev/api/people/6/",
    "https://swapi.dev/api/people/7/",
    "https://swapi.dev/api/people/8/",
    "https://swapi.dev/api/people/9/",
    "https://swapi.dev/api/people/10/",
    "https://swapi.dev/api/people/12/",
    "https://swapi.dev/api/people/13/",
    "https://swapi.dev/api/people/14/",
    "https://swapi.dev/api/people/15/",
    "https://swapi.dev/api/people/16/",
    "https://swapi.dev/api/people/18/",
    "https://swapi.dev/api/people/19/",
    "https://swapi.dev/api/people/81/"
    ],
"planets":[
    "https://swapi.dev/api/planets/1/",
    "https://swapi.dev/api/planets/2/",
    "https://swapi.dev/api/planets/3/"
    ],
"starships":[
    "https://swapi.dev/api/starships/2/",
    "https://swapi.dev/api/starships/3/",
    "https://swapi.dev/api/starships/5/",
    "https://swapi.dev/api/starships/9/",
    "https://swapi.dev/api/starships/10/",
    "https://swapi.dev/api/starships/11/",
    "https://swapi.dev/api/starships/12/",
    "https://swapi.dev/api/starships/13/"
    ],
"vehicles":[
    "https://swapi.dev/api/vehicles/4/",
    "https://swapi.dev/api/vehicles/6/",
    "https://swapi.dev/api/vehicles/7/",
    "https://swapi.dev/api/vehicles/8/"
    ],
"species":[
    "https://swapi.dev/api/species/1/",
    "https://swapi.dev/api/species/2/",
    "https://swapi.dev/api/species/3/",
    "https://swapi.dev/api/species/4/",
    "https://swapi.dev/api/species/5/"
    ],
"created":"2014-12-10T14:23:31.880000Z",
"edited":"2014-12-20T19:49:45.256000Z",
"url":"https://swapi.dev/api/films/1/"
```

CRUD relates to REST

CRUD Operation	**Entity already exists**	**Entity does not already exists**
CREATE	Error	POST or PUT
READ	GET	Error
UPDATE	PUT or PATCH	Error
DELETE	DELETE	Error

Curl Example:

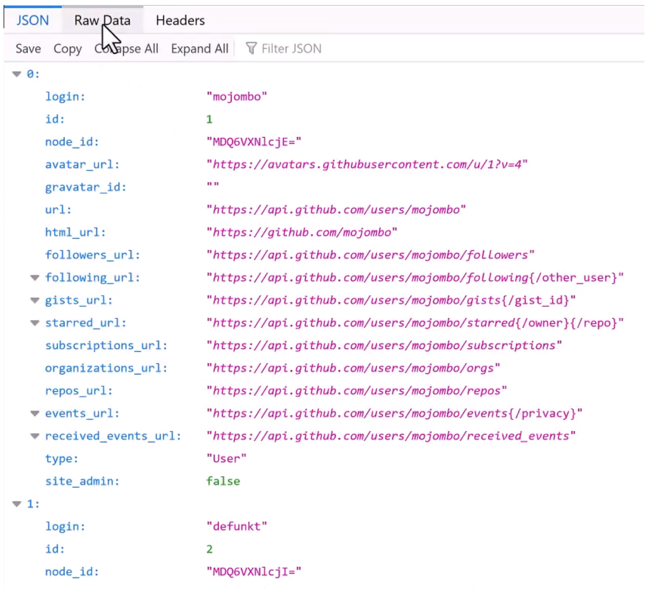
- 404 NOT Found error as entry is not available in the database



```
{"detail":"Not found"}riggy@riggy-Latitude-5310:~/Desktop$ curl https://swapi.dev/api/films/7 -v
*   Trying 52.58.110.120:443...
* Connected to swapi.dev (52.58.110.120) port 443 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
*  CAfile: /etc/ssl/certs/ca-certificates.crt
*  CApath: /etc/ssl/certs
* TLSv1.0 (OUT), TLS header, Certificate Status (22):
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.2 (IN), TLS header, Certificate Status (22):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS header, Certificate Status (22):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (IN), TLS header, Certificate Status (22):
* TLSv1.2 (IN), TLS handshake, Server key exchange (12):
* TLSv1.2 (IN), TLS header, Certificate Status (22):
* TLSv1.2 (IN), TLS handshake, Server finished (14):
* TLSv1.2 (OUT), TLS header, Certificate Status (22):
* TLSv1.2 (OUT), TLS handshake, Client key exchange (16):
* TLSv1.2 (OUT), TLS header, Finished (20):
* TLSv1.2 (OUT), TLS change cipher, Change cipher spec (1):
* TLSv1.2 (OUT), TLS header, Certificate Status (22):
* TLSv1.2 (OUT), TLS handshake, Finished (20):
* TLSv1.2 (IN), TLS header, Finished (20):
* TLSv1.2 (IN), TLS header, Certificate Status (22):
* TLSv1.2 (IN), TLS handshake, Finished (20):
* SSL connection using TLSv1.2 / ECDHE-RSA-AES128-GCM-SHA256
* ALPN, server accepted to use h2
* Server certificate:
*  subject: CN=swapi.dev
*  start date: Aug  1 07:25:19 2023 GMT
```

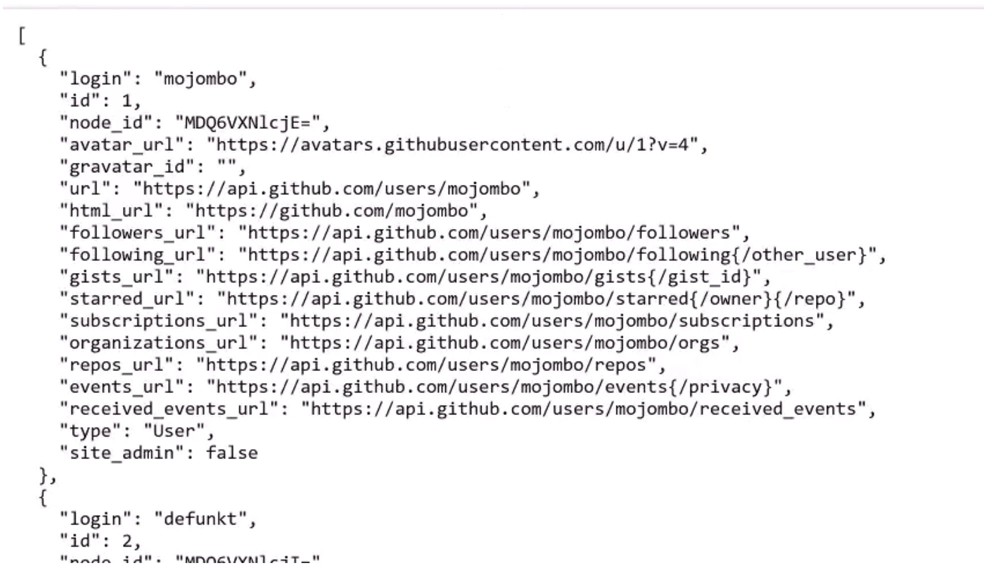
GitHub GET Example

- https://api.github.com/users
- Returns users of github in JSON format



```
▼ 0:
    login:                "mojombo"
    id:                   1
    node_id:              "MDQ6VXNlcjE="
    avatar_url:           "https://avatars.githubusercontent.com/u/1?v=4"
    gravatar_id:          ""
    url:                  "https://api.github.com/users/mojombo"
    html_url:             "https://github.com/mojombo"
    followers_url:        "https://api.github.com/users/mojombo/followers"
  ▼ following_url:        "https://api.github.com/users/mojombo/following{/other_user}"
  ▼ gists_url:            "https://api.github.com/users/mojombo/gists{/gist_id}"
  ▼ starred_url:          "https://api.github.com/users/mojombo/starred{/owner}{/repo}"
    subscriptions_url:    "https://api.github.com/users/mojombo/subscriptions"
    organizations_url:    "https://api.github.com/users/mojombo/orgs"
    repos_url:            "https://api.github.com/users/mojombo/repos"
  ▼ events_url:           "https://api.github.com/users/mojombo/events{/privacy}"
  ▼ received_events_url:  "https://api.github.com/users/mojombo/received_events"
    type:                 "User"
    site_admin:           false
▼ 1:
    login:                "defunkt"
    id:                   2
    node_id:              "MDQ6VXNlcjI="
```

- Raw Data shows JSON array
- https://api.github.com/users/Yuvraj-26 returns specific user
- https://api.github.com/users?per_page=2 returns two user results
- https://api.github.com/users?per_page=2&since=1005 returns two users results starting from entry 1005 where id is 1006



```
[
  {
    "login": "mojombo",
    "id": 1,
    "node_id": "MDQ6VXNlcjE=",
    "avatar_url": "https://avatars.githubusercontent.com/u/1?v=4",
    "gravatar_id": "",
    "url": "https://api.github.com/users/mojombo",
    "html_url": "https://github.com/mojombo",
    "followers_url": "https://api.github.com/users/mojombo/followers",
    "following_url": "https://api.github.com/users/mojombo/following{/other_user}",
    "gists_url": "https://api.github.com/users/mojombo/gists{/gist_id}",
    "starred_url": "https://api.github.com/users/mojombo/starred{/owner}{/repo}",
    "subscriptions_url": "https://api.github.com/users/mojombo/subscriptions",
    "organizations_url": "https://api.github.com/users/mojombo/orgs",
    "repos_url": "https://api.github.com/users/mojombo/repos",
    "events_url": "https://api.github.com/users/mojombo/events{/privacy}",
    "received_events_url": "https://api.github.com/users/mojombo/received_events",
    "type": "User",
    "site_admin": false
  },
  {
    "login": "defunkt",
    "id": 2,
```

GitHub POST Example

- Post is used to create an entry into a database that's managed through an API
- Personal access token provide access to API
- Curl client to create a repo on Github using access token
- Returns JSON structured data

Postman API Platform to test API

- Create a new collection
- New Request
- Select POST request and specify request URL
- HEADERS
    - Accept header for which content types the client is able to understand
    - Authorisation header used to provide credentials that authenticate a user agent with a serve
- BODY
    - Specify raw data in JSON format
    - Add Content
- Send HTTP request and we get back HTTP response in JSON format
- New repository created in GItHub
- Create collections and automate collections for testing APIs
- Postman application can be installed locally to test APIs on local machine
- HTTP status code is 201 Created

Alternative utility is https://restfox.dev/

HTTP Response codes in REST APIs

- 200 OK: This status code indicates that the request was successful, and the server has returned the requested resource. It is the most common status code for successful HTTP requests.

- 201 Created: This status code is used to indicate that the request has been successfully processed, and a new resource has been created as a result. It is typically used in POST requests to create new resources.

- 400 Bad Request: This status code is returned when the server cannot process the client's request due to invalid syntax or parameters in the request. It indicates that the client's request is malformed and should be corrected.

- 401 Unauthorized: This status code is returned when the client's request lacks proper authentication credentials or the provided credentials are invalid. It indicates that the client needs to provide valid authentication to access the requested resource.

- 404 Not Found: This status code indicates that the requested resource could not be found on the server. It is returned when the URI provided in the request does not match any known resources.

- 405 Method Not Allowed: This status code is returned when the client attempts to use an HTTP method (e.g., POST, GET, PUT, DELETE) that is not supported for the requested resource. It informs the client that the chosen HTTP method is not allowed for the resource in question.

- 500 Internal Server Error: This status code is a generic error message indicating that something went wrong on the server while processing the request. It is used when the server encounters an unexpected condition that prevents it from fulfilling the request.