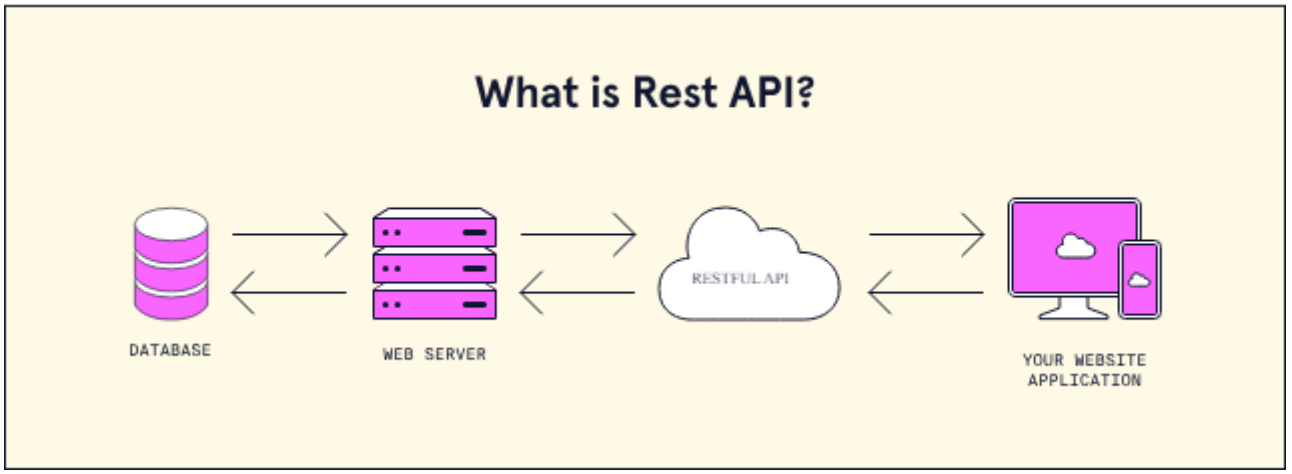
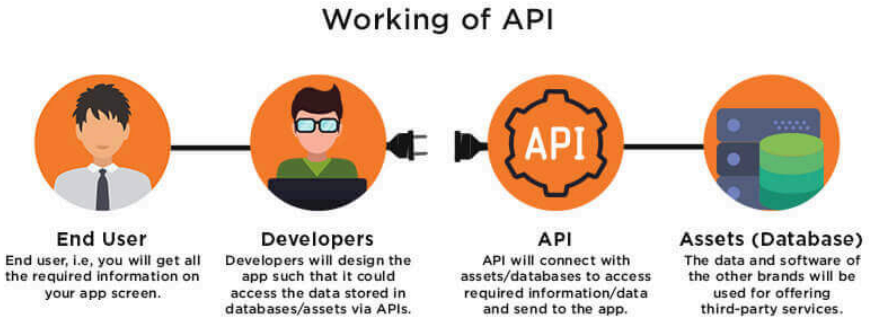
REST: Representational State Transfer

- Way of creating a Standard for communication between systems eg. CLIENT – SERVER (or server – server, databases etc.)
- "A server will respond with the representation of a resource which contains links that can be followed to make the state of the system change which in turn makes a new request and expect another response with more representations of the resource being requested (resource is often HTML or JSON document for example)"
- Client and Server are decoupled / separation of concerns.
- Systems are stateless - meaning neither need to know what state the other is in. This is enforced using resources rather than commands and does not rely on implementation of interfaces. Server/Client can understand messages received without having context of previous messages



API: Application Programming Interface

- Interface/Contract of Communication connecting computers/software to each other (clients, servers)
- Set of protocols that allow communication and exchange of data.
- The syntax/code specification used/implemented between a service provider and the computer/software making the request.
- API specification defines calls - how to call, use, implement API methods/requests/endpoints etc.

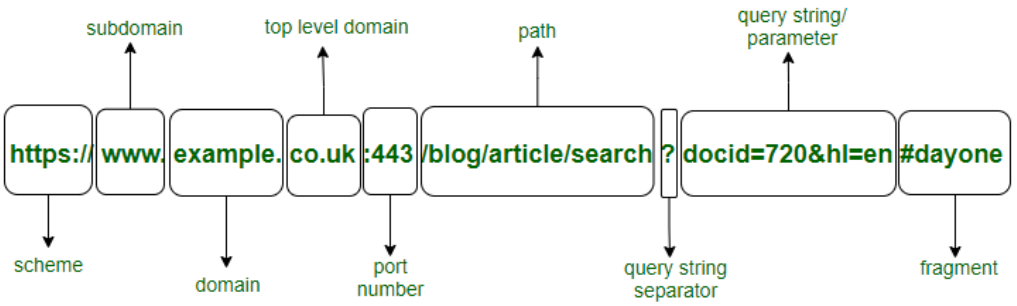


Deno runtime (instead of Node)

- JS/TS, WebAssembly runtime environment and package manager (doesn't need npm)
- Lots of packages/modules built-in compared to node
- Web standard APIs -uses web standard APIs whenever possible, which makes your code more portable and future-proof.
- Security focused (no access to files/network/environment unless explicitly granted),
- Modern Architecture - modern JavaScript features and web standards, making it a future-proof choice for development.

DNS, IP Addresses & Ports

URL: Uniform Resource Locator



- Can have components like query parameters, ports, anchors etc. depending on what you are trying to locate.

DNS - Domain Name System:

- Takes domain names, uses DNS servers to return IP addresses (computers can only work with numbers not strings) to route data between clients, servers
- <terminal> **nslookup** google.com
    - (nameserver) nslookup is a network administration command-line tool for querying the Domain Name System to obtain the **mapping between domain name and IP address**, or other DNS records.
    - https://www.nslookup.io/

IP Address:

- What the network, computers, servers, switches, routers etc. use as reference to communicate between systems, route data/packets, communicate.
- Two kinds: IPv4 - 32 bit, IPv6 - 128 bit

Network Ports:

- Ports on routers and switches (physical)
- Computers have **Virtual Ports (16 bits)**
    - Apps on computer use one port each to ‘listen’ and stream data independent of each other
    - Reserved ports (<1024) for things like mail, FTP, HTTP, HTTPS etc. (should not be used) https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers

Example: DNS → gets IP Address → HTTP req to IP_Address:port

Network Protocols

- DNS, IP addresses, ports - ways to identify, get to destination/source
- Protocols are the rules of communication (across a network)
- Protocols are the actual communication mediums/system of rules that allow communication and transmission of data between systems (define rules, syntax, semantics, formats, synchronization, error methods etc.)

LAYER	APPLICATION/ EXAMPLE		CENTRAL DEVICE PROTOCOLS			DOD4 MODEL
APPLICATION (7) Serves as the window for users and application processes to access the network services.	End User Layer: Program that opens what was sent or creates what is to be sent		**User Applications** SMTP		GATEWAY	Process
PRESENTATION (6) Formats the data to be presented to the Application layer. It can be viewed as the "Translator" for the network.	Syntax Layer: Encrypt & decrypt (if needed)		JPEG/ASCII/EBDIC/ TIFF/GIF/PICT			
SESSION (5) Allow session establishment between processes running on different stations.	Synch & send to ports (logical ports)		**Logical Ports** RPC/SQL/NFS/ NetBIOS names			
TRANSPORT (4) Ensures that messages are delivered error-free, in sequence, and with no losses or duplications.	TCP: Host to Host, Flow Control	PACKET FILTERING	TCP/SPX/UDP			Host to Host
NETWORK (3) Controls the operations of the subnet, deciding which physical path the data takes.	Packets: "letter", contains IP address		**Routers** IP/IPX/ICMP			Internet
DATA LINK (2) Provides error-free transfer of data frames from one node to another over the Physical layer.	Frames: "envelopes", contains layer 2 address (ex MAC address)		**Switch Bridge WAP** PPP/SLIP	LAN Based Layers		Network
PHYSICAL (1) Concerned with the transmission and reception of the unstructured raw bit stream over the physical medium.	Physical structure: Cables, hubs, etc.		**Hub**			

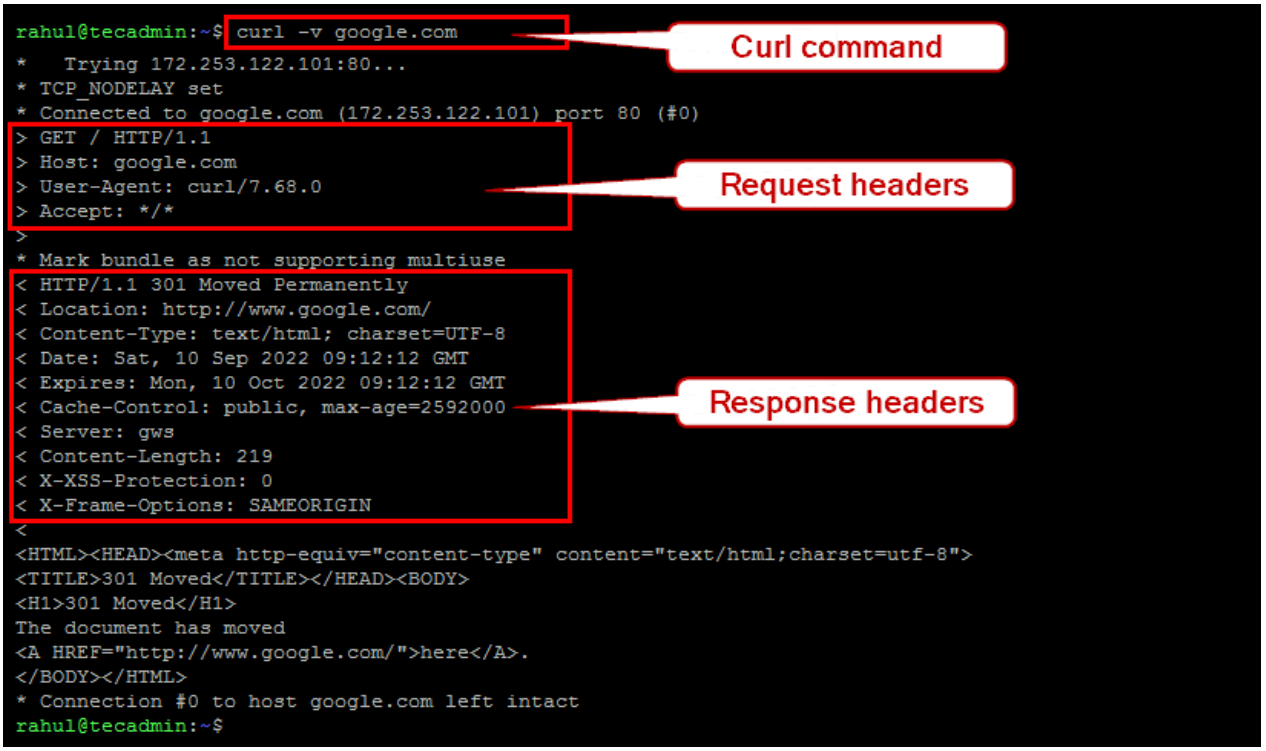
OSI Model -

- [7, 6, 5] layers - imp. Protocols include HTTP(S), TLS, IMAP (email), DNS, FTP (file transport)
- [4] Transport layer also includes secure transfer mechanisms
- [2] Data Link Layer eg. Ethernet, WLAN
- [1] Physical Layer eg. Fiber optic cables, Coaxial cables
- As you go up layers there is increasing Abstraction eg. at lower layers it is binary code running through cables for example. The higher layers can work/use the appropriate lower level layers

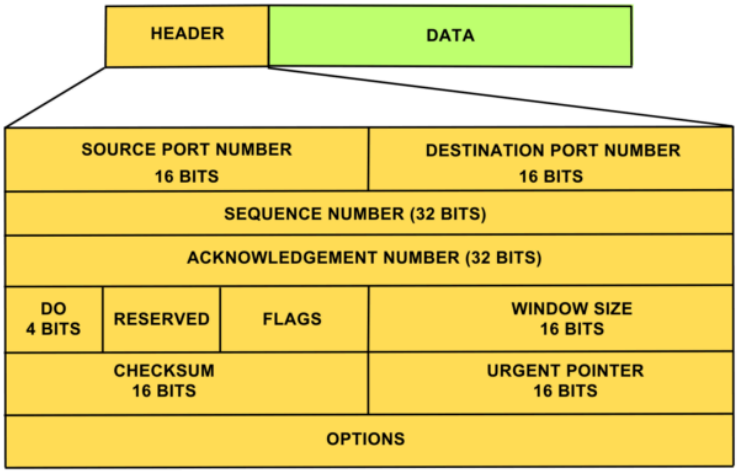
HTTP (default port 80):

- HyperTextTransferProtocol - Designed to transfer information between networked devices
- Separate connection for each request message - response resource/function/message (HTTP 1.1)
- Reuse single connection for multiple req-res (HTTP 2). This reduces a lot of overhead, speeds up connections.
- HTTPS: HTTP over TLS - 'secure' version of HTTP that uses TransportLayerSecurity protocol for encryption and authentication of websites via certificates / public-private key encryption (port 443)
    - Compared to HTTP which communicates using unencrypted plain text that can be read/manipulated if intercepted/listened to

- HTTP Headers:
    - Standardised info, transferred along with req or res in packets, with specific layout format containing data eg. Req headers might have info about the resource being requested for or the client. Res headers might have info like the location of the resource or the server providing it. Payload headers might include info like the encoding used.

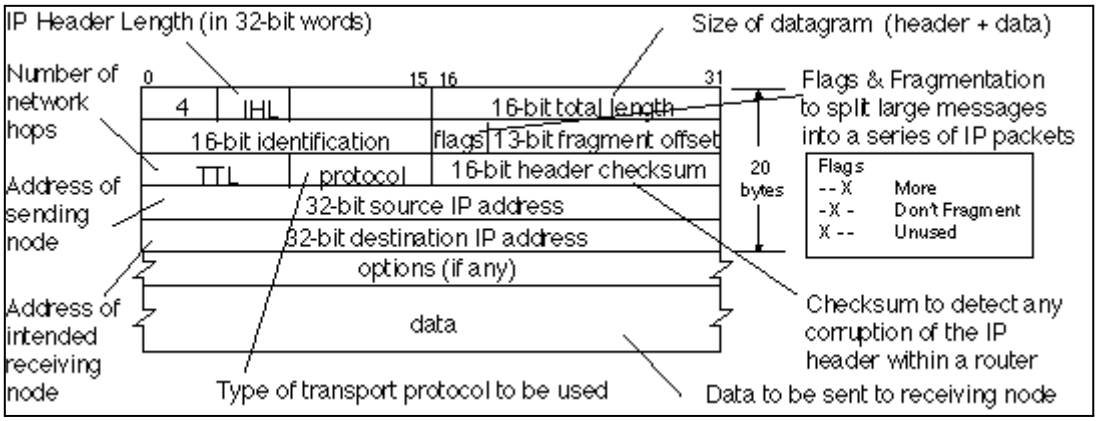


- TCP: Transmission Control Protocol:
    - Extremely common transport layer protocol. Way to send message packets over a network.
    - Used to transmit reliable, ordered, error-checked data delivery(eg using checksums).
    - Usually uses IP (Network layer below transport layer) to deliver data over networks
    - 1. Client GET req made → 2. SEND server res → client verifies data → 3. server expects GOT msg sent back that will be sent back to verify data received/transmitted correctly (acknowledge)
    - Compare header to HTTP header:
        - Deals specifically with the ports, IP deals with the addressing part
        - Includes sequence and acknowledgement number
        - Checksum for error-checking that bits have been hashed correctly



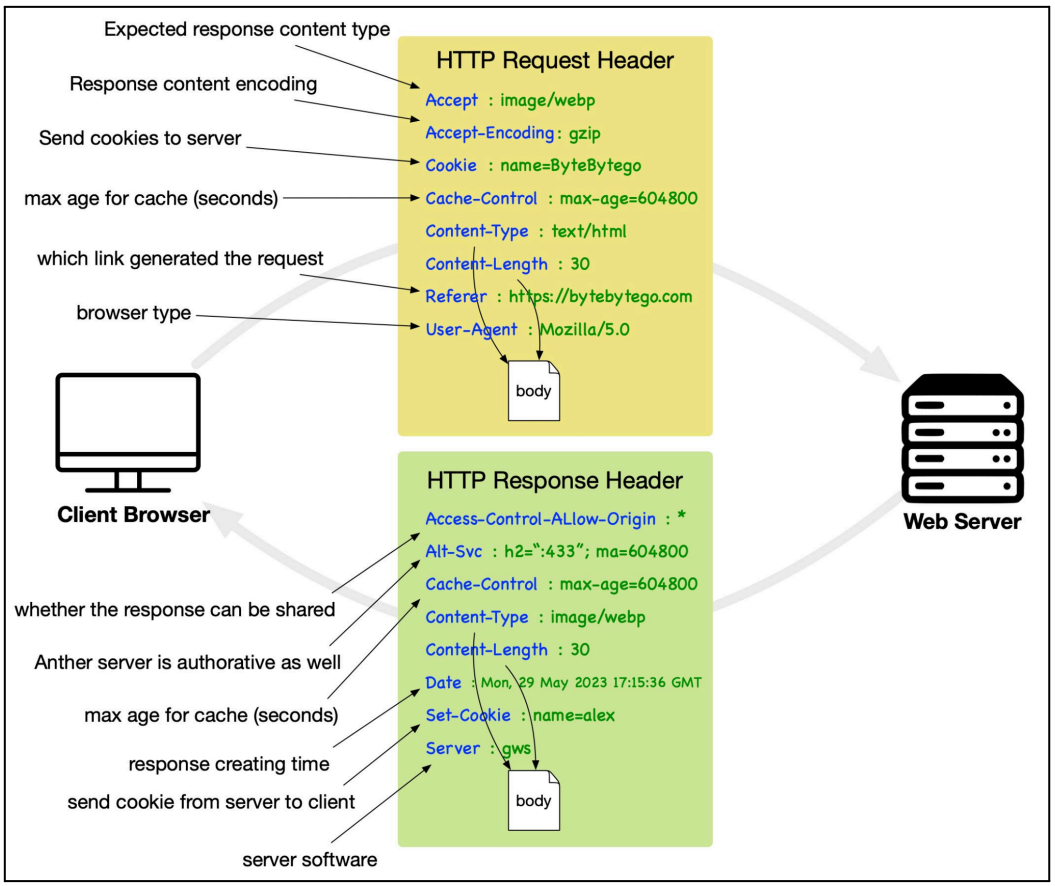
- UDP: User Datagram Protocol:
    - Main alternative to TCP.
    - Simple, faster, real-time data delivery as it doesn't include acknowledgement step.
    - Streams packets in chunks quickly so used for media a lot
    - UDP Header only has ports, length, checksum and data

- IP: Internet Protocol:
  - Network layer communication protocol the way to deliver packets from source to destination
  - Works with other protocols in upper layers (eg. TCP/IP)
  - Header has source and destination IP addresses (TCP provides the ports)



HTTP Headers and Methods

- Allows metadata to be transmitted (additional info about the data). Think of <meta> in <head> of an html document.
- Request headers sent by client with the req so server listening has info relevant to how it responds
- Response headers sent by server sent with res back to client with info such as status, content-length/type, compression info.
- Can be accessed through req, res object and is a standard API



Headers in Fetch:

- Fetch API has a Header Web API that can be attached to req/res and can be accessed/manipulated using code - Example create headers (objects) in JS and attach to fetch req

HTTP Request Methods:

- What type of operation to perform (GET, POST, PUT, PATCH, DELETE, HEAD (only headers for route/resource sent in res without body), OPTIONS (query options for methods/supported operations that can be used with API, also does not include data body))l;p

HTTP Status Codes

When a browser request a service from a web service, a response code will be given.
These are the list of HTTP Status code that might be returned

1XX Information	
100	Continue
101	Switching Protocols
102	Processing
103	Early Hints
2XX Success	
200	OK
201	Created
202	Accepted
203	Non-Authoritative Information
205	Reset Content
206	Partial Content
207	Multi-Status (WebDAV)
208	Already Reported (WebDAV)
226	IM Used (HTTP Delta Encoding)
3XX Redirection	
300	Multiple Choices
301	Moved Permanently
302	Found
303	See Other
304	Not Modified
305	Use Proxy
306	Unused
307	Temporary Redirect
308	Permanent Redirect
4XX Client Error	
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
Compiled by Ivan Tay.	

4XX Client (Continue)	
407	Proxy Authentication Required
408	Request Timeout
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Payload Too Large
414	URI Too Large
415	Unsupported Media Type
416	Range Not Satisfiable
417	Exception Failed
418	I'm a teapot
421	Misdirected Request
422	Unprocessable Entity (WebDAV)
423	Locked (WebDAV)
424	Failed Dependency (WebDAV)
425	Too Early
426	Upgrade Required
428	Precondition Required
429	Too Many Requests
431	Request Header Fields Too Large
451	Unavailable for Legal Reasons
499	Client Closed Request
5XX Server Error Responses	
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout
505	HTTP Version Not Supported
507	Insufficient Storage (WebDAV)
508	Loop Detected (WebDAV)
510	Not Extended
511	Network Authentication Required
599	Network Connect Timeout Error

Creating a server

Request interface of the fetch API represents a resource request
Create new request object with Request( ) constructor
https://developer.mozilla.org/en-US/docs/Web/API/Request

The Response interface of the Fetch API represents the response to a request.
You can create a new Response object using the Response() constructor
https://developer.mozilla.org/en-US/docs/Web/API/Response

- MIME types
  - **Used in 'content-type' header** - format is: type/subtype
  - Specifies to browser/client type of data so it knows how to handle the data.
  - Reference list - https://www.iana.org/assignments/media-types/media-types.xhtml
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types
  - https://en.wikipedia.org/wiki/Internet_Assigned_Numbers_Authority
    - The Internet Assigned Numbers Authority (IANA) is a standards organisation that oversees global IP address allocation, autonomous system number allocation, root zone management in the Domain Name System (DNS), media types, and other Internet Protocol–related symbols and Internet numbers.

Server Side Routing:
Lots of libraries and frameworks (eg. express) for routing
Explore how to do it with just deno, js/ts and web standard APIs

- Conditionally route req sent to servers which might use different logic + send different res depending on what route client what requesting (eg. /login vs /forum
- Separate Server code/logic/functions

Ways we can route:
(We are assuming that we will route using the Request.url)

- if/else statements, switch statements, Regex,
  [built into deno] URLPattern API (matches URLs or parts of URLs against a pattern. The pattern can contain capturing groups that extract parts of the matched URL.)
  https://developer.mozilla.org/en-US/docs/Web/API/URLPattern/URLPattern

Server Side Data Storage

How to store data/allow for persistent data (remain accessible):
Data storage options -

- In-memory (eg. Objects, Maps, Arrays) - [as long as program is running you have access to the variable and its value - variable exists in the JS code when script is running]
  - Fast, quick to setup and use; easy access to data(eg. indexing)
  - Not persistent; doesn't scale beyond one script/app/server; limited memory
- Text file (eg. txt, JSON, CSV)
  - Fast, easy to set up and use; easy access to data (eg. read file from disk/storage location); data persists in the file
  - Strict formatting of data to match file allowed data types; doesn't scale beyond one script/app/server (file exists in one place, difficult to synchronise file if script running on multiple computers)
- Key-Value storage (eg. LocalStorage, Redis) - similar to objects but format/structure is strict
  - Similar to Maps/Objects in-memory storage except we store in short-term storage
  - Easy to access data; we can have persistent data depending on service (eg.Redis - acts as a cache for data before or after it goes to a database)
  - Suitable to use mainly for simple data in short formats/data stored for short-term solution; Relational data requires a lot of management/maintenance

- The more In-memory/text-files/k-v storage tools you use the more complex and difficult to manage/analyse/store/back-up data and updating the app as you use more tools. (consider that you are keeping data stored in different formats/places etc.)
- Relational database eg. SQL
    - Robust, consistent, long-term, highly structured data storage solution. Multiple backups can be maintained
    - Works at scale - Vertical/Horizontal scaling solutions
    - SQL used to query/analyse data quickly, easily
    - More effort to set up and maintain; require rigid data schemas to define format
- Object/non-Relational Database (eg. MongoDB)
    - Robust, consistent, long-term, **unstructured** data storage solution. Multiple backups can be maintained
    - Much easier to scale compared to relational db - Works at scale - Vertical/Horizontal scaling solutions
    - Uses a different query language (not SQL) to query/analyse data quickly, easily
    - Medium effort to set up and maintain; https://deno.land/x/sqlite@v3.8
    - More relaxed when it comes to data schemas to define format - they aren't necessary however need to be aware of schemas 'drifting' over time where the app becomes buggy because you haven't maintained a strict standard about data format that is allowed to be stored in it.
- Other storage tools:
    - Graph databases: Allow data to be stored in visual relationships format
    - Column based databases, vector databases

GET Requests

- One of the HTTP methods used to request server/backend for resources/content
- Read-only requests (CRUD) - no manipulation/changing of the data
- Importance of the URL structure so it can be used to send data to/from the API so we know what data we are getting.

REST API design

Importance of design architecture/hierarchy of url:

- Route paths & name for resource/API (nouns - use plural or singular eg. do you want all users or a specific userId)
- Logical/hierarchy of endpoint nesting (eg. posts/postId/comments) vs tradeoffs with search params - need to consider relationships of the data/search param/API
- Allow for filtering,nesting, number sequencing/pagination/paging (to break lists of content such as search results, blog posts, product listings)
- Create Error handling logic - also informing client about what might have gone wrong + you need to log errors to address possible issues
- Document and keep record of API + updated with examples