Database → servers with long life that interact with the rest of your app thru network calls, with protocols on top of TCP or even HTTP

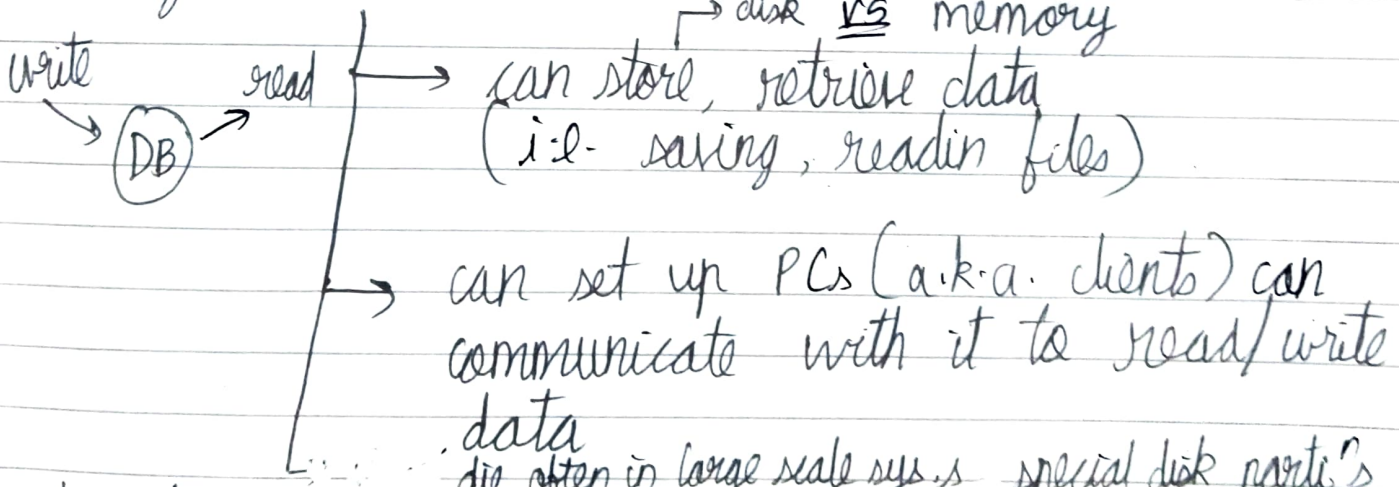
# STORAGE

→ 1) Every system requires storage.
eg:- for storin user info, metrics

→ 2) Database = a machine that helps storing and retrieving data

synonymous word pairs w.r.t. data:
- setting ≡ writing ≡ recording ≡ storing
- getting ≡ reading ≡ querying ≡ retrieving

In most cases database (DB) is actually just a server
eg:- even a PC can be made to act as a DB




a.k.a volumes are used by DB to access volumes of data
die often in large scale sys.s special disk partn's even if machine dies (crashes permanently)

→ 3) Persistance of data in a DB
↳ non-volatile storage → can retrieve info after being power cycled

Disk - writin data to disk persists even if the DB itself crashes / faces some outage

eg:- excludin extreme issues, any file you save on PC, persists even after PC shuts down or crashes.

↳ needs const. power to retain data

↳ RAM - volatile storage

Memory - Data stored in this does not persist if DB goes down

eg:- any data stored in variable in your server's DB does not persist if DB server goes down

eg:- In a mobile, say 64GB storage, 8GB RAM

↳ 64GB → disk

↳ 8GB → memory

NOTE Reading data from memory is faster than reading data from disc.

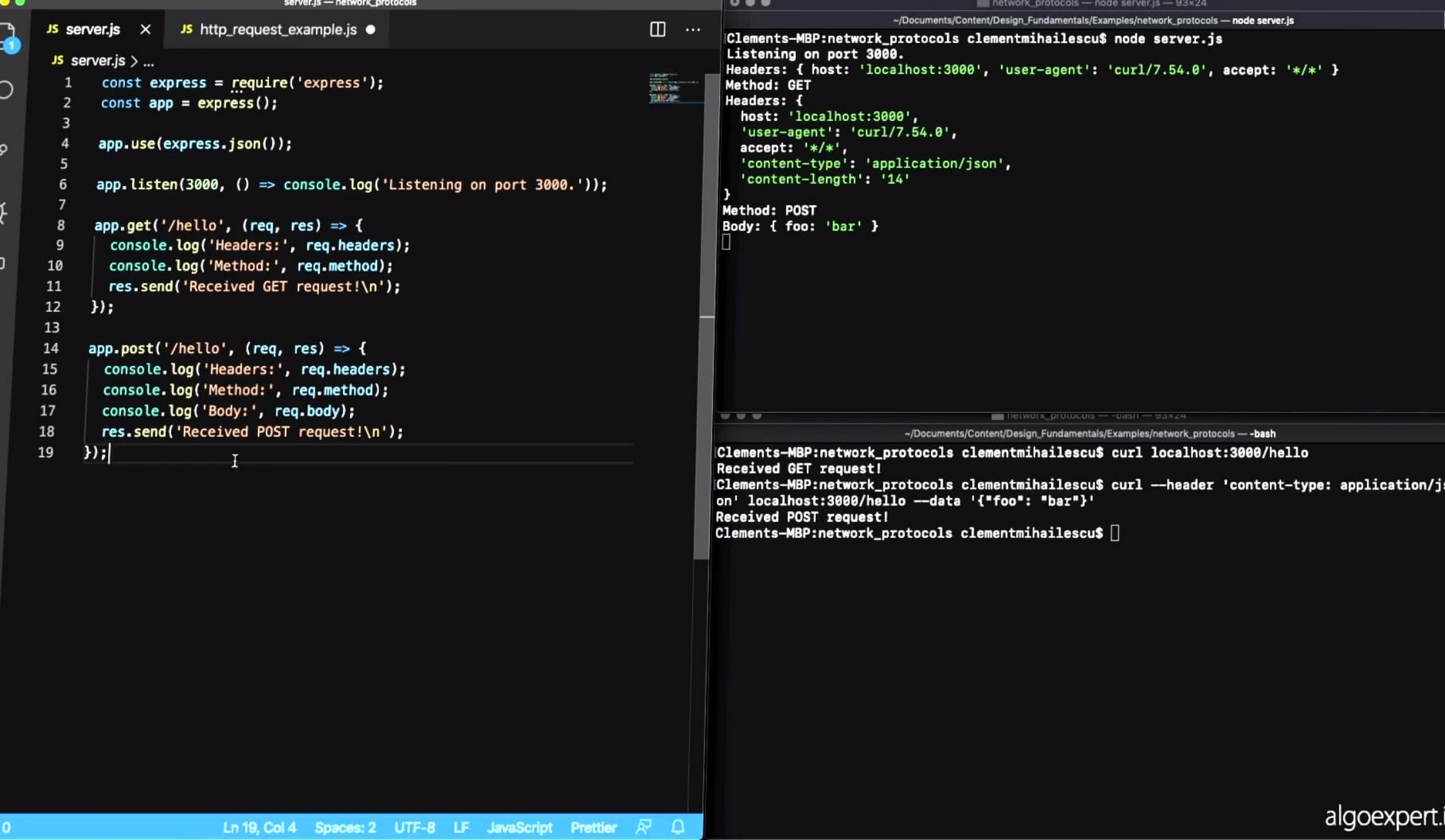
There are a lotta DB offerings to give options based on performance, data security

When DB goes down, ~~because~~ depending on how critical a part of the entire sys the DB is, does the system crash or persist through

Distributed storage → storing data on multiple machines.

↳ M-1 → split data up

↳ M-2 → replicate data on machines

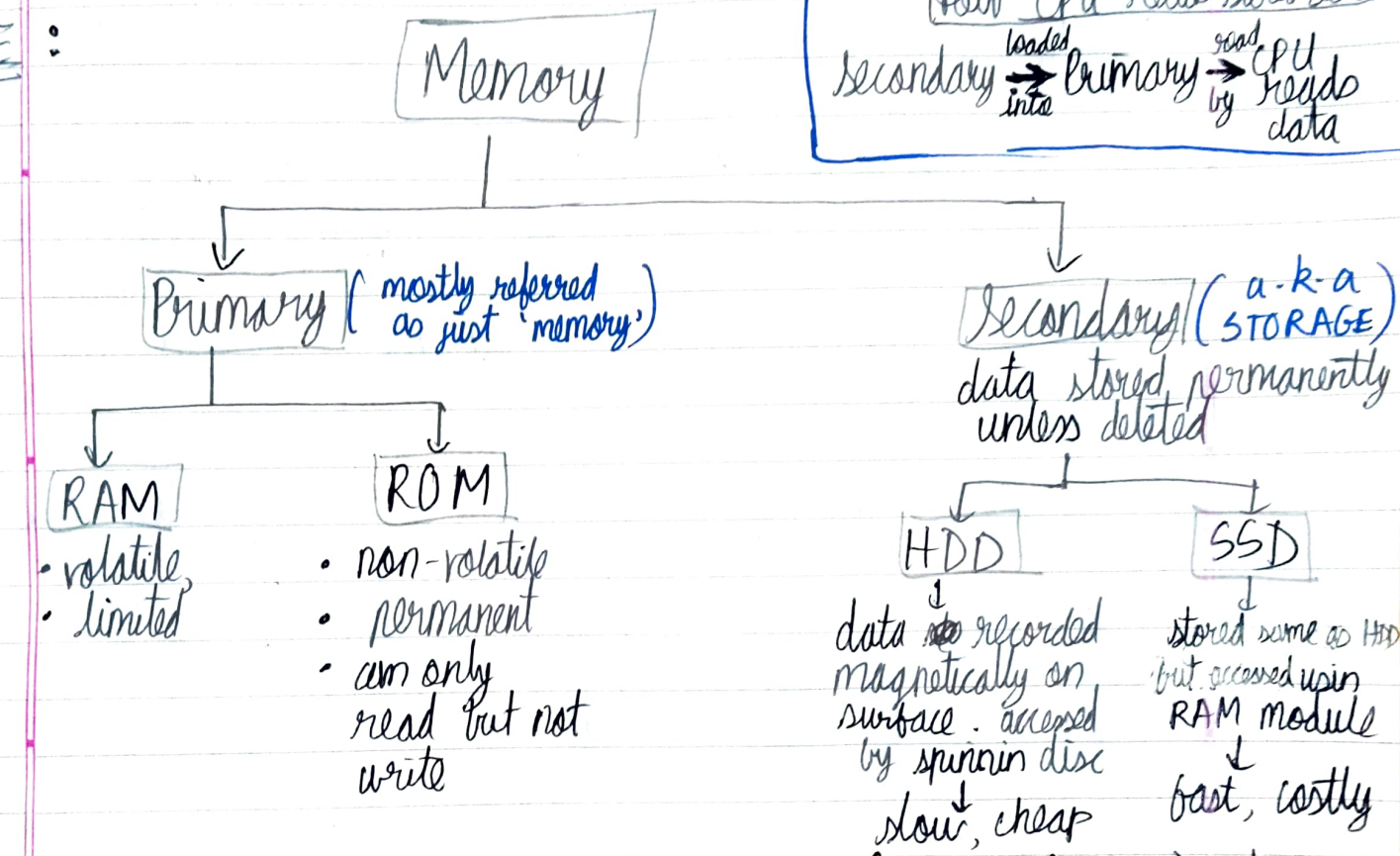

server.js — network_protocols
server.js
http_request_example.js
server.js > ...
1 const express = require('express');
2 const app = express();
3
4 app.use(express.json());
5
6 app.listen(3000, () => console.log('Listening on port 3000.'));
7
8 app.get('/hello', (req, res) => {
9 console.log('Headers:', req.headers);
10 console.log('Method:', req.method);
11 res.send('Received GET request!\n');
12 });
13
14 app.post('/hello', (req, res) => {
15 console.log('Headers:', req.headers);
16 console.log('Method:', req.method);
17 console.log('Body:', req.body);
18 res.send('Received POST request!\n');
19 });
Ln 19, Col 4
Spaces: 2
UTF-8
LF
JavaScript
Prettier
network_protocols — node server.js — 93×24
~/Documents/Content/Design_Fundamentals/Examples/network_protocols — node server.js
Clements-MBP:network_protocols clementmihailescu$ node server.js
Listening on port 3000.
Headers: { host: 'localhost:3000', 'user-agent': 'curl/7.54.0', accept: '*/*' }
Method: GET
Headers: {
host: 'localhost:3000',
'user-agent': 'curl/7.54.0',
accept: '*/*',
'content-type': 'application/json',
'content-length': '14'
}
Method: POST
Body: { foo: 'bar' }
network_protocols — -bash — 93×24
~/Documents/Content/Design_Fundamentals/Examples/network_protocols — -bash
Clements-MBP:network_protocols clementmihailescu$ curl localhost:3000/hello
Received GET request!
Clements-MBP:network_protocols clementmihailescu$ curl --header 'content-type: application/j
on' localhost:3000/hello --data '{"foo": "bar"}'
Received POST request!
Clements-MBP:network_protocols clementmihailescu$
algoexpert.

**Consistency** - A concept in storage referring to the staleness or up-to-dateness of data

eg:- on accessing data from a DB, how fresh/up-to-date is the data that you get

How CPU reads stored data:
Secondary → (loaded into) Primary → (read by) CPU reads data

NOTE :




- ROM contains a prog. called BIOS (Basic I/O sys) which microprocessor (i.e. computer CPU) to load OS from HDD into RAM whenever PC is turned on. Newer motherboards use UEFI (Unified Extensible Firmware Interface)

- Analogy :- (i) MEMORY → a work desk
  (ii) storage → Secondary memory → desk drawers where files are stored
  (iii) memory → Primary memory → table top of desk
  (iv) Prog in memory → tools, things on desk (easy, fast to access)
  (v) Prog in storage → files in desk drawers that you gotta put on table top to access (slow to access)
  (vi) Loading a prog. from secondary memory → open a drawer, remove required file and put it on table top to use it