A screen shot of a computer

Description automatically generated<https://www.youtube.com/watch?v=W2Z7fbCLSTw>

7 Database Paradigms

1. Key-value (Redis)
   1. Javascript Object
   2. Stored in RAM not Disk
      1. A screenshot of a computer

         Description automatically generatedPro: Fast
      2. Cons: Limited memory, no queries
   3. Used for caching
2. Wide Column
   1. Like a relational but doesn’t have a schema
   2. Can scale horizontally
   3. Good for time series, historical records (Netflix storying history)
   4. A diagram of a document

      Description automatically generatedNo joins (?)
3. Document (MongoDB, Firestorm, DynamoDB?)
   1. Document is a container for key, value pairs
   2. Unstructured and doesn’t require a schema
   3. Grouped together inside a collection
   4. Cons: doesn’t support joins, so instead of normalizing your data, encourage to embed the data into a single document. Pro: reads from a front end app are much faster but writing/updating is more complex
   5. Not ideals for graphs (social networking apps)
4. Relational (MySQL, postgres)
   1. Inspired the development of SQL (Structured Query Language)
   2. Foreign keys, because they reference data in a different table
   3. A screenshot of a computer

      Description automatically generatedOrganizes data in its smallest normal form
   4. Requires a schema
   5. ACID (Atomicity, …)

SQL vs NoSQL

<https://www.youtube.com/watch?v=_Ss42Vb1SU4>

SQL

-Structured Query Language

-Table Based

-Vertical Scaling (increase size of instance)

-Relational

* allows easy querying on relationships between data amount multiple tables

-Data is structured

* therefore potential error is reduced because SQL schemas require the data model and format of the data to be known before storying anything

-ACID compliant

* Atomicity (all or nothing transactions)
* Consistency (Data is valid before or after)
* Isolation (multiple transaction at the same time)
* Durability (committed data is never lost)

-SQL transaction are groups of statements that are executed atomically (meaning all or nothing if any statement in the group fails)

Cons

-Structure (columns/tables) must be created in advance

-Not effective for storing and querying unstructured data

-Difficult to scale horizontally because of their relational nature. For read heavy systems its straightforward to provision multiple read only replicas but write heavy systems usually the option is to vertically scale the database up which is generally more expensive than provisioning additional servers

noSQL

-document, key value, graph, or wide column

-Horizontal Scaling (adding more instances)

-More flexible and easier to setup

-Data sharding (technique where a large dataset is divided into smaller), makes it more easier to horizontal scaling

Cons

-Loss of consistency

-Eventual Consistency

Comments:

-SQL can scale horizontally well

Horizontally scaling: can add more servers or nodes to DB cluster to handle increased data volume, traffic

Vertical Scaling: adding more CPU, RAM or storage capacity to a single server to handle increased demands

-MongoDB is ACID after version 4?

A black screen with white text

Description automatically generated

Environment Variables

-UPPER\_SNAKE\_CASE

-Provide different values for different environments

-Store secrets, such as database URLs and credentials or API keys

Ex

-In a prod env, may want additional logging and analytics compared to dev env

-Building an API connected to a db but want to use a separate test db. Can pass a test db’url and cred when develop locally

NODE\_ENV  
-common env variable

Loading Env variables

-One way is to define the env var and their values directly in command to run your code

NODE\_ENV=prod VIDEO\_URL="https://www.youtube.com/watch?v=X2CYWg9-2N0" node index.js

-But this might get cumbersome quickly if you have a lot of env var

-If you had sensitive data like database credentials, that’s even worse since you wouldn’t want to push your package.json if it contained those values in an npm script!

Export

-Instead of setting env var directly in the command, you can use shell command (export) which will save env var to the current shell session (new shell would not have access since a it’s a new env)

export NODE\_ENV=prod VIDEO\_URL="https://www.youtube.com/watch?v=X2CYWg9-2N0"

Copy

printenv

-to view all env var

-but shows too much, just want to see app env var.

-also we lose our env var when we terminate the shell

dotenv

-one of the most common ways

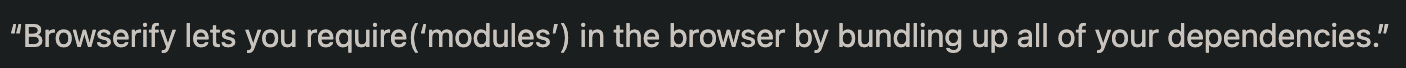
-create a file called .env in the root that will contain all env var

-must be added to .gitignore

-then import dotenv

-BUT projects where a whole team needs synced access to the same env var may benefit from robust and flexible options

<https://peterxjang.com/blog/modern-javascript-explained-for-dinosaurs.html>



A screen shot of a computer program

Description automatically generated

Old way if you wanted to use a library (code that someone else wrote)

A screenshot of a computer

Description automatically generated

A black background with white text

Description automatically generated



-How we used to make website with JS libraries. Easy to understand but downloading new updates was a pain

-JS package managers helped to automate the process of downloading and upgrading libraries from a central repository

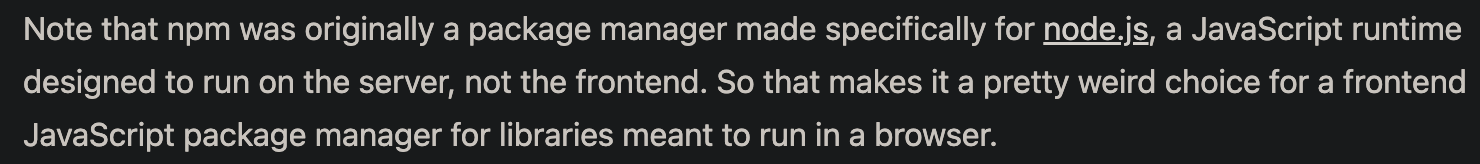
npm (JS package manager) 2015

-yarn 2016, uses npm under the hood

npm

-was a package manager for node.js (JS runtime designed to run on the server, not frontend)

-package managers generally uses command line



-Have node.js then already have npm

A black background with white letters

Description automatically generated

-on your index.html

-generate package.json (configuration file, that uses npm to save all project info)

A computer screen shot of a program code

Description automatically generated

A screen shot of a computer code

Description automatically generated



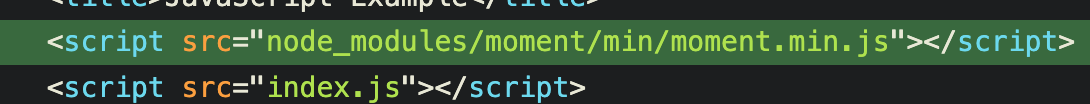
-to install moment.js JS package

-downloads all code from moment.js into a folder called node\_modules

-modifies the package.json file to keep track of project dependency

-This is great because the node modules can be very large, just need to share package.json and other developers can use npm install

-Means we no longer have to use manually download from website and we can download and update is using npm



-Bad thing is we are digging through node modules folder to find the location of each package, and manually include it in HTML

-Next we will look to automate

Webpack (Javascript module bundler)

-JS was designed to only run in the browser, with no access to file system of the client’s computer (for security reasons)

-Required you to load each file with variables shared globally (Example: moment) ,

-

Node