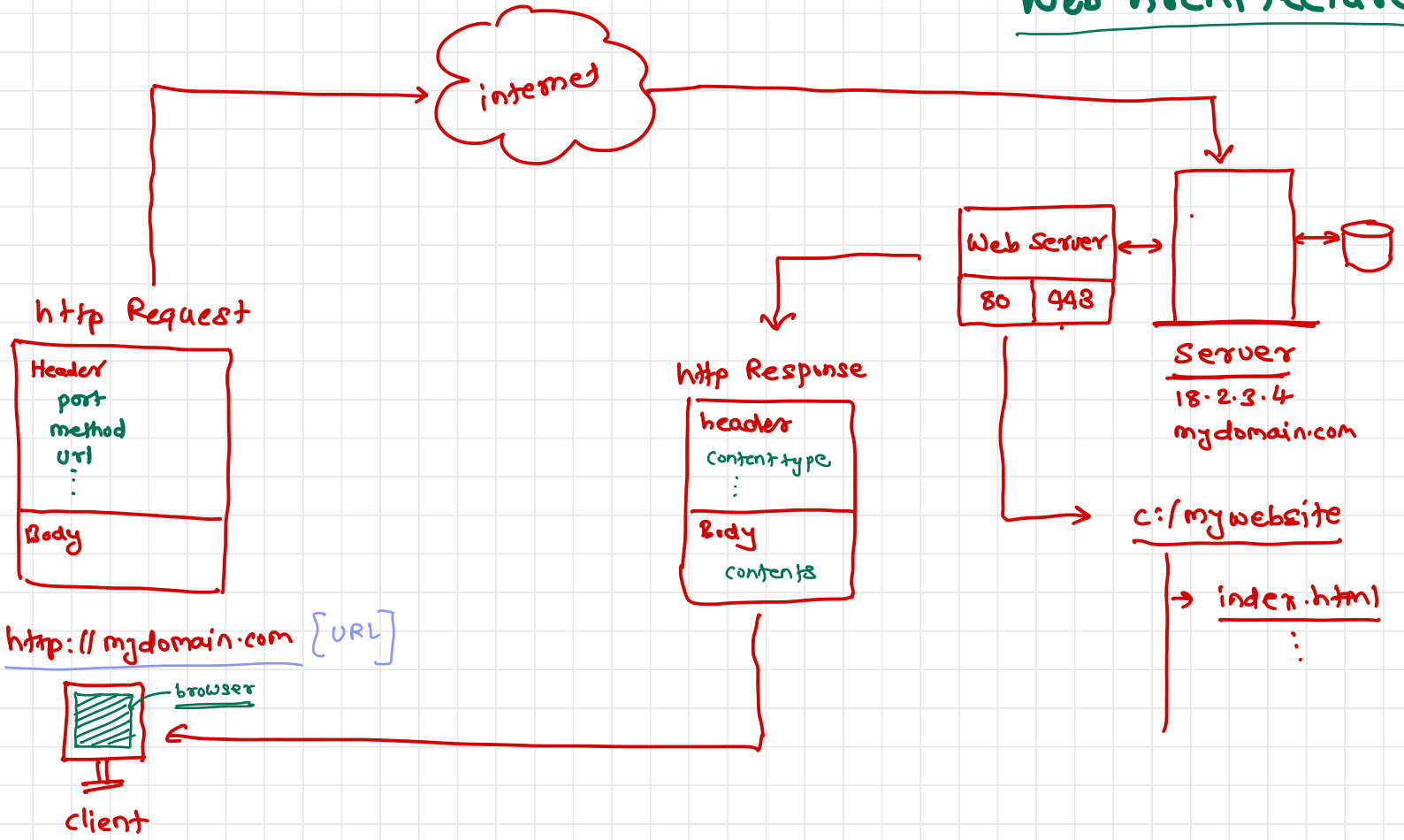
JavaScript

JavaScript

About your instructor

- 14+ years of experience
- Associate Technical Director at Sunbeam
- Worked in various domains using different technologies
- Developed
  - 180+ mobile applications on iOS and Android platforms
  - Various websites using PHP, MEAN and MERN stacks
  - Various machine learning solutions (using Python)
- Certification completed
  - Certified Kubernetes Application Developer (CKAD)
  - Certified Kubernetes Administrator (CKA)
  - Certified Jenkins Engineer (CJE)
  - Docker Certified Associate (DCA)

Web Architecture

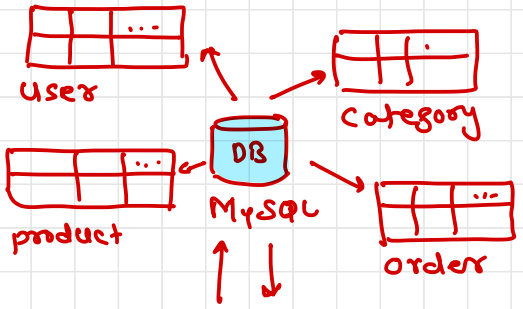


method	operation
POST	insert into ....
GET	select ..from...
PUT/PATCH	update ....
DELETE	delete from...

REST

POST /order
GET /product
Update /user

Operation	Method
Create	POST
Read	GET
Update	PUT, PATCH
Delete	DELETE



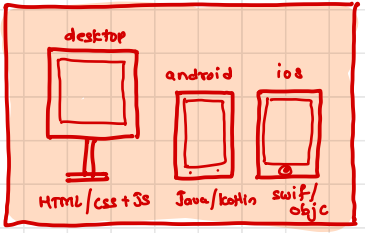
Web Server

Backend

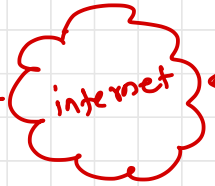
Request
- path
- method

Response
JSON / XML

frontend



Client



User

Frontend
- visible to user
- provides backend connectivity
- also known as
  - client
- technologies
  - html/css
  - JS
    - Angular
    - React
    - VueJS
    - NextJS *
    - jQuery
  - python
    - Django

Backend
- DB connectivity
- design patterns
  - REST
  - GraphQL *
- invisible to end user
- also known as
  - middleware
  - web server
  - web services
- technologies
  - Java EE
  - JS → Express, Fastify, hapi
  - TS → NestJS
  - PHP → CodeIgnitor, Laravel
  - Python → Flask,

full stack developer
- MEERN
- MEAN
- WISA
- LAMP

Frontend

- React *
- Angular
- PHP
- Vue
- Java
- Python Django

Backend

- TypeScript *: NestJS * ...
- Express, Fastify, hapi
- JavaEE
- PHP
- .Net

Database

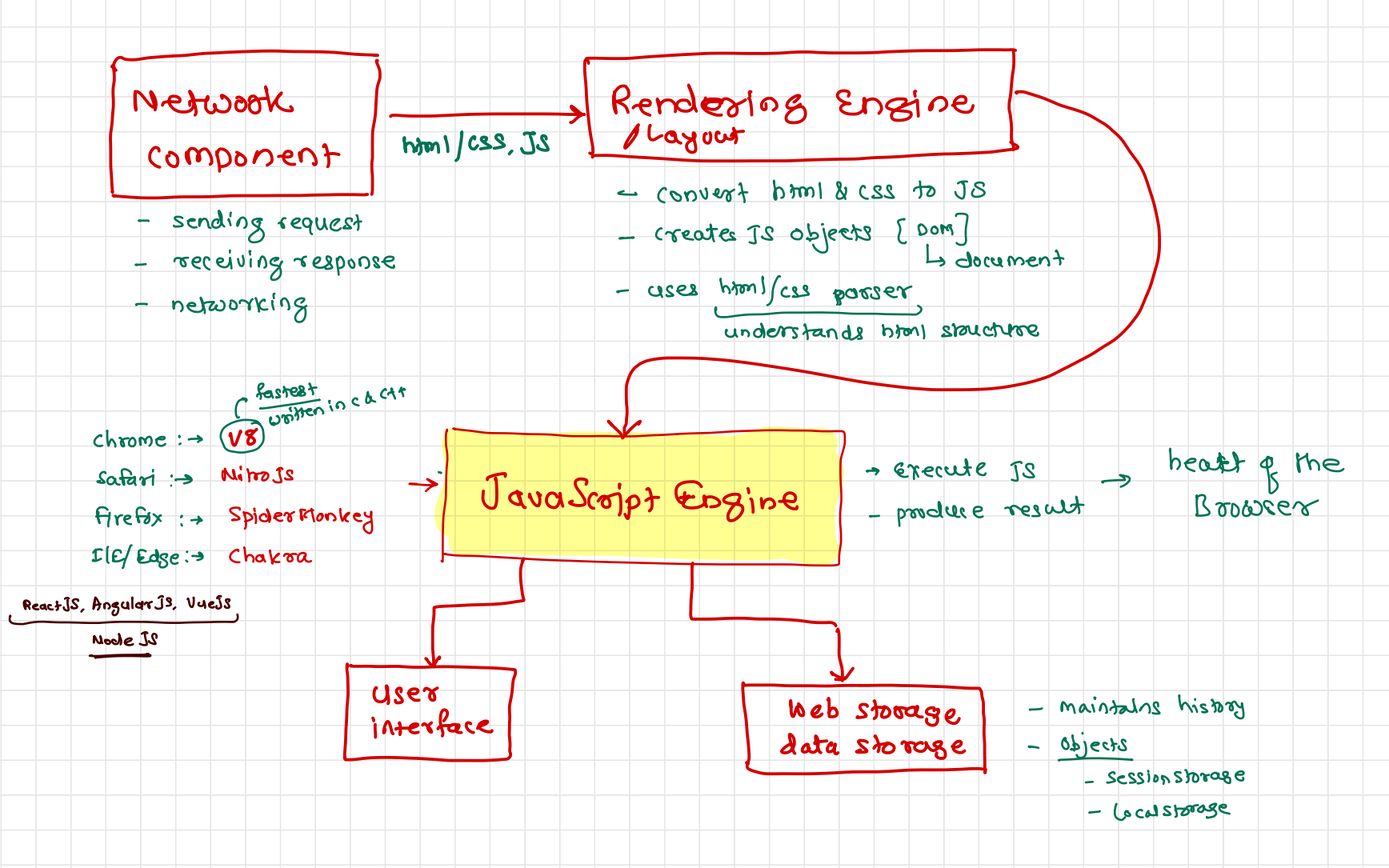
- NoSQL
  - mongo *, firebase
- RDBMS
  - oracle
  - PostgreSQL
  - MySQL
  - MS SQL Server

URL

- uniform resource Locator
- input to the browser
- carries information to detect & browse webpages
- https://google.com
- components
  - scheme :- protocol used in the URL
    - http, https, ftp, local ....
  - domain name / IP address :- of server machine
  - Port number : http(80), https(443)
  - path / file name : file name to be browsed
  - query string : input given to the page (http://mydomain.com/index.php?name=sunbeam)
    - key = name
    - value = sunbeam
    - query string = name=sunbeam
  - hash component : used to link multiple sections within a page

Response

- will be sent by server
- response may contain html/json/xml/text/images
- contains
  - → Header
    - ① status code -
      - for client to check the status of operation performed
        - 1xx : debugging messages
        - 2xx : success, (201, 202, 200...)
        - 3xx : migration (301, 302, ...)
        - 4xx : client error (401, 403, 404)
        - 5xx : server error (500, 502)
    - ② content-type
      - → type of content
        - text/plain, text/html, application/json


Network component
- sending request
- receiving response
- networking
html/css, JS
Rendering Engine
/Layout
- convert html & css to JS
- creates JS objects [DOM]
↳ document
- uses html/css parser
understands html structure
Chrome :→ V8
fastest
written in c & c++
Safari :→ Nitro JS
Firefox :→ SpiderMonkey
IE/Edge :→ Chakra
ReactJS, AngularJS, VueJS
Node JS
JavaScript Engine
→ execute JS
- produce result
heart of the Browser
User interface
Web storage
data storage
- maintains history
- objects
- sessionStorage
- localStorage

Introduction

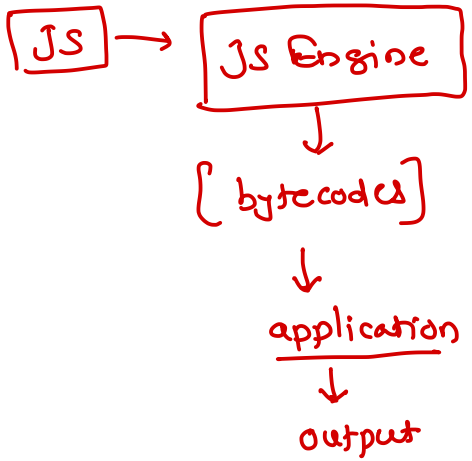
Overview

- JavaScript was initially created Brendan Eich to make the web pages alive
- The earlier name of JS was LiveScript
- The programs written in this language are called as scripts
- They can be written in the web page to execute at the time of page load
- Browser can execute the JS code because it has a built-in engine called a “JavaScript Virtual Engine”

JavaScript Execution Engine

: used for executing JS

- It is a program or an interpreter which executes JS code
- A JS engine can be implemented as a standard interpreter or just-in-time compiler that compiles the JS code to bytecode in some form
- There are many implementations of JS engines
  - V8 – Open source developed by Google and written in C++
  - SpiderMonkey – developed by Mozilla Foundation and today powers Firefox
  - Chakra – developed by MS for IE (JScript) and Edge (JS)
  - JavaScriptCore – also known as Nitro, developed by Apple for Safari
  - Rhino – managed by Mozilla and written in Java
  - KJS – developed for KDE's Konqueror browser
  - JerryScript – is a lightweight engine for IoT



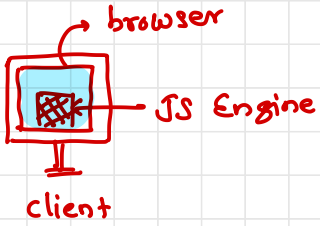
JavaScript Today

- Today JS programs can be executed not only in the browsers but also on the server or on any device that has a JavaScript engine running
- Applications
  - Used in Web pages [using <script>]
  - Business applications (using TypeScript in Angular, React, Vue)
  - Utility Applications (console scripting)
  - Game development (Unity)
  - Mobile application development (Cordova, React-Native, Nativescript)
  - Server side development (Express)
    - used to communicate with DB

database : mongoDB, mysql,
server : Express / fastify
client :
- desktop : React / Angular / PHP
- mobile :
  - ios : swift / RN
  - android : Java / kotline / RN

Client side

- html / CSS : designing
- JS : client side programming



Server side

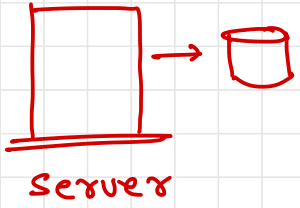
Stack : language + Web server + DB + platform

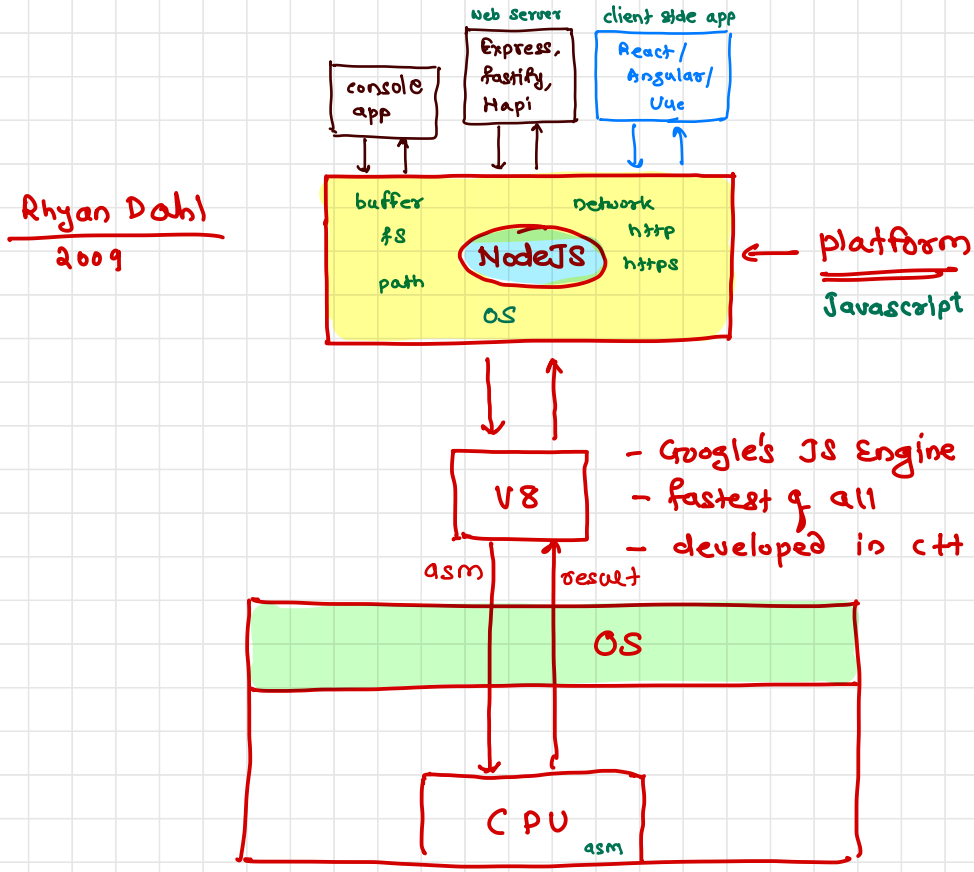
① AMP : Apache + MySQL + PHP / Perl / Python

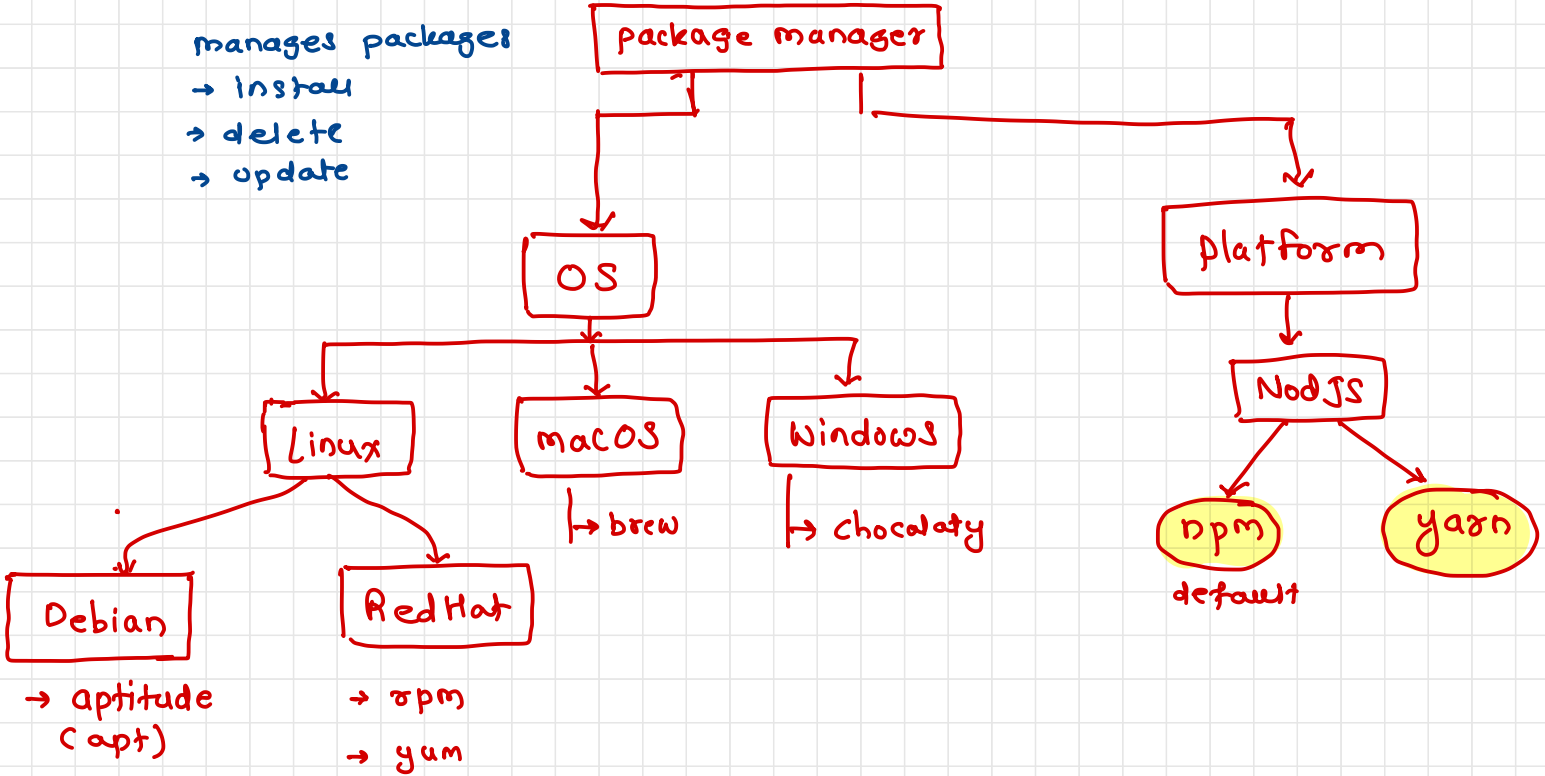
LAMP = Linux + AMP
MAMP ⇒ macOS + AMP
WAMP ⇒ Windows + AMP

② WISA :- Windows + IIS + SQL server + ASP.net

③ MERN / MEAN ⇒ Mongo / MySQL + Express + React / Angular + Node




Rhyan Dahl
2009
console app
Web server
Express, fastify, Hapi
client side app
React / Angular/ Vue
buffer
fs
path
network
http
https
NodeJS
OS
platform
Javascript
V8
- Google's JS Engine
- fastest of all
- developed in c++
asm
result
OS
CPU
asm


manages packages
→ install
→ delete
→ update
package manager
OS
platform
Linux
macOS
Windows
NodJS
→ brew
→ chocolaty
npm
yarn
default
Debian
RedHat
→ aptitude (apt)
→ rpm
→ yum

> npm init

> npm install <package>

- local : used to install packages used programmatically
  - gets installed in current directory
- global : - used to install utilities [yarn, ... nodemon, pm2]
  - use --global or -g

yarn

→ npm install -g yarn

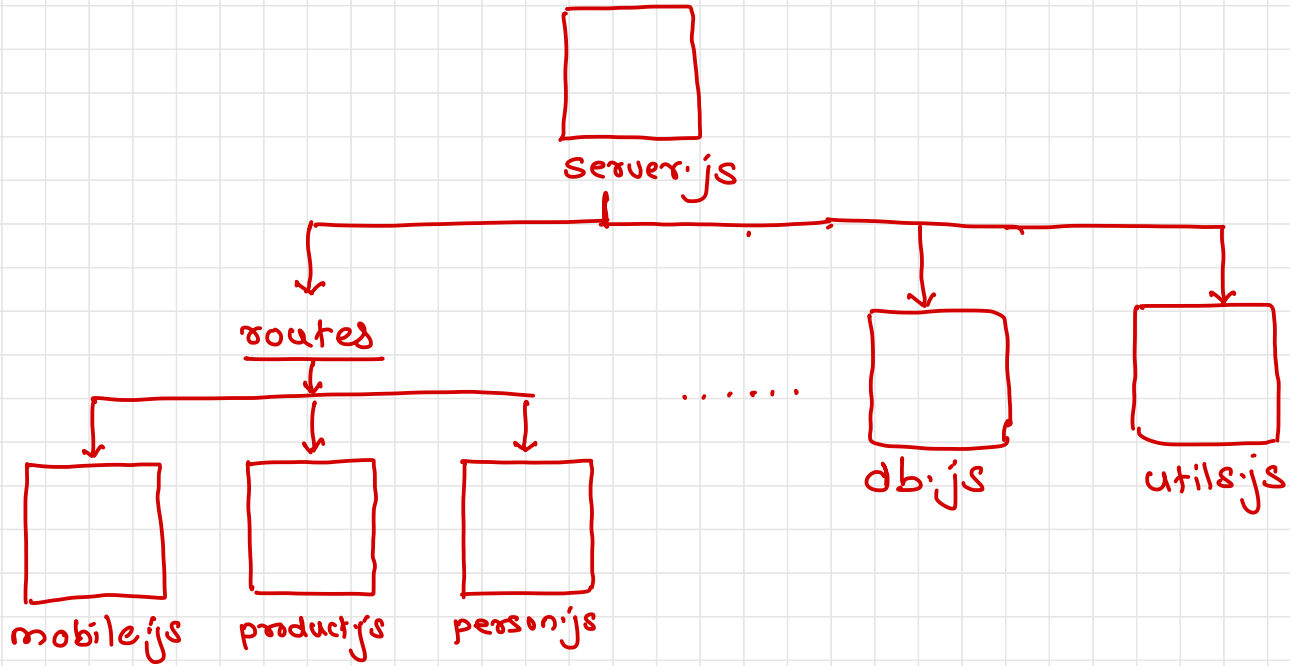
npm	yarn
> npm install <..>	> yarn add <..>
> npm init	> yarn init
> npm install	> yarn

To send input to the server

- query string
  - http://localhost:4000/product?title=product2&price=200
    - keys
    - values
  - use `request.query`
  - values are optional

- path parameters
  - http://localhost:4000/product/4
    - 4 → path parameter
  - use `request.params`
  - value is mandatory

- request body → use `request.body`
  - body is available with all methods except GET
  - use when
    - you want pass many params
    - data is non-ascii (binary)
    - data size is huge


server.js
routes
. . . . . .
db.js
utils.js
mobile.js
product.js
person.js

```
app = express()

app.get('/person',...)

app.post('/mobile',...)
    :
```

Router

path	callback
GET /	—
GET /person	—
POST /mobile	—

Routing table

GET /person → valid response (200) — route found

POST /category → invalid response (404) — route does not exist

Fundamentals

Features

- Case sensitive name, Name, HostName FirstName
- Semicolon is optional if every statement is written on separate line
- Code can be commented using // (one line) or /* */ (multiline)

Built-in objects

- console : prints in developer console
- window : shows visually

Variables

- A variable is a named storage for data
- We can store any value of any type in a variable
- Variables are mutable
- JavaScript provides a keyword **let** to declare a variable
- Syntax
  - let <varname> = <value>
- e.g.
  - let age = 30
  - let firstName = 'steve'
  - let email = "person@test.com"
  - let address = `
    A-100, Near Sunbeam,
    Pune, 411000
    `

Constants

- Variables which do not change the values
- Constants are immutable
- Preferred over variables
- JS provides a keyword **const** to declare a constant
- Syntax
  - const <name> = <value>
- e.g.
  - const pi = 3.14
  - const gst = 18

Data Types

- In JavaScript, all data types are inferred
- Automatically set by JS by checking the ”current” value in the const or variable
- JS provides a operator **typeof** to check the variables datatype [can be used as a function as well]

Data Types

- **Number**
  - both whole and decimal numbers
  - Can not represent integer values larger than 2^{53} or less then -2^{53}
- **BigInt**
  - Represent big number
  - Uses ‘n’ at the end
- **String**
  - Represents string values
  - Can use single quote, double quote or back quote
- **Boolean**
  - Represents logical value
  - Can have only true or false
- **null**
  - Special value
- **undefined**
  - Special value represents a variable which does not have any value
- **Objects**
  - Objects represents collection of key value pairs

Type Conversions

- Used for converting value from one type to another
- String conversion
  - Uses String() to convert to string
- Number conversion
  - Number() to convert to number
  - parseInt() to covert to whole number
  - parseFloat() to convert to decimal number
- Boolean conversion
  - Uses Boolean() to covert to boolean

NaN → Not a Number

Number ('twenty')

Comparison

- Equality (==)
  - Checks only value and ignores the data type
  - E.g.
    - “20” == 20 => true
- Identity/strict equality (===)
  - Checks both value and data type
  - E.g. “20” === 20 => false

Array (collection)

- Array represents collection of values
- e.g.
  - const numbers = [10, 20, 30, 40, 50]
  - const countries = [“india”, “usa”, “uk”]

Functions

- Block of organized, reusable code that is used to perform single, related action
- JavaScript provides **function** keyword to declare a function
- Syntax

```
function greet() {
        console.log("welcome to JS")
}
```

- Types
  - Parameter less function
  - Parameterized function

Objects

Overview

- Object represents collection of properties and methods
- In JS, object can be created and modified dynamically [properties can be added dynamically]
- JS provides following ways to create objects
  - Object Literals
  - Constructor functions
  - Classes

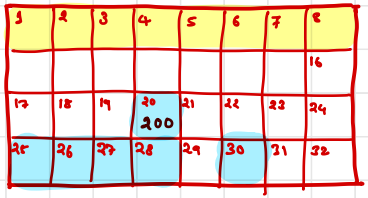
Object Literals

JSON :- JS object Notation

- Simplest way to create objects
- Uses {} to enclose the key-value pairs
- E.g.
  - const person = {
    - firstName: 'Steve',
    - lastName: 'Jobs
  - }

values

keys/properties



mutable

```
let num = 100
```

immutable

```
const num = 100
```

0x20

100

num = 200

0x20

~~100~~ 200

immutable

```
const person = {
    name: "xyz",   address: "pune" }
```

0x30

0x25

person (reference)

0x25

```
name: "xyz"
address: "pune"
```

Object

```
person.name = "abc"
person.address = "mumbai"
```

✓

```
person = { name: "abc", ... }
```

X

as person is immutable

Object Literals

- Object literals can use property shorthand
- E.g.

```
function registerUser(firstName, lastName) {
        const person = {
                firstName, lastName
        }
}

registerUser('steve', 'jobs')
```

Object Properties

- Properties are used to store data inside object
- Properties are key-value pairs
- Properties can be added by using
  - [] syntax
  - Dot (.) syntax
- E.g.
  - person['age'] = 30
  - person['email'] = 'person@test.com'

Inspecting Object properties

- JS provides Object.keys() to inspect the object properties
- You can also use for..in loop to get all the properties of an object
- E.g
  - const p = {name: ‘person1’, address: ‘pune’}
  - console.log(Object.keys(p))

  - for (let property in p) {
    - console.log(property)
  - }

Property Descriptor

- Property descriptor can be used to get the information about the property
- E.g.
  - const p = {name: ‘person1’, address: ‘pune’}
  - cosole.log(Object. getOwnPropertyDescriptor(p, ‘name’))

- Modifying property
  - Property can be modified by using property descriptor
  - To make the property read only
    - Object.getOwnPropertyDescriptor(person, ‘firstName’, {writable: false})
  - To skip the property from enumerating
    - Object.getOwnPropertyDescriptor(person, ‘firstName’, {enumerable: false})
    - Non-enumerable properties will not be serialized
  - To lock a property from modification
    - Object.getOwnPropertyDescriptor(person, ‘firstName’, {configurable: false})

Property Getter and Setter

- Getter
  - Also known as inspector
  - Method used to return value
  - E.g.

```
Object.defineProperty(p, 'fullname', {

    get: function() {
        return
          `${this.firstName} ${this.lastName}`
    }

})
```

- Setter
  - Also known as mutator
  - Method used to modify value
  - E.g.

```
Object.defineProperty(p, 'fullname', {

    set: function(value) {
        const parts = value.split(' ')
        this.firstName = parts[0]
        this.lastName = parts[1]
    }

})
```

Using Constructor function

- Generally preferred when multiple objects are required of similar type
- Constructor function is the function which is used for creating new objects
- Generally written starting with capital letter
- Should be executed using **new** keyword
- e.g.

```
// using Object constructor function
const person = new Object()

// using custom function
function Person(name, address) {
        this.name = name
        this.address = address
}
```

Function Prototype

- A function prototype is the object instance that will become the prototype for all the objects created using the function as a constructor
- To access the function prototype use prototype prototype property on the function
- E.g.

```
function Car(model, company) {
        this.model = model
        this.company = company
}

console.log(Car.prototype)
```

Object Prototype

- An object prototype is the object instance from which the object is inherited
- To access the object prototype use __proto__ prototype
- E.g.

```
function Car(model, company) {
        this.model = model
        this.company = company
}

const c1 = new Car('i20', 'Hyundai')
console.log(c1.__proto__)
```

Prototype under the hood

```
function Car(model, company) {
        this.model = model
        this.company = company
}

Car.prototype.price = 5

const c1 = new Car('i20', 'hyundai')
const c2 = new Car('nano', 'tata')

c1.price = 7.5

console.log(c1.price)
console.log(c2.price)

Car.prototype.price = 3
```

Class

- JS supports classes in the latest version of ECMA
- You need latest versions of all the browsers to run this code
- NOTE: IE does not support JS classes
- Benefits of using classes
  - Makes the coding simpler
  - Similar to other languages
  - Preferred using constructor function or object literals
- Syntax

```
class <name> {
    // class body
}
```

Adding constructor

- No matter which class you are writing, a constructor in JS always have same name **constructor**
- Like other languages it will be called automatically
- Unlike other language you can have only one constructor in the class
- E.g.

```
Class Person {
        constructor() {
                // constructor body
        }
}
```

Adding methods

- Method is a function inside a class
- A method does not require any function keyword for declaration
- A method can be parameterless or parameterized
- Unlike other languages
  - JS does not provide any kind of access specifiers
  - JS does not support method overloading
- E.g.

```
class Person {
        ….
        canVote() {
                return this.age >= 18
        }
}
```

Adding getter and setter

- Getters and setters are used to set or get values of properties
- **get** and **set** keywords are used to declare them
- E.g.

```
Class Person {
        …

        get fullName() {
                return `${this.firstName} ${this.lastName}`
        }

        set fullName(name) {
                ….
        }
}
```

Inheritance

- JS uses **extends** keyword to extend one class [subclass] from another [super]
- Like other languages, subclass inherits all the properties and methods from super class
- E.g.

```
class Student extends Person {
        ....
}
```

Method overriding

- Changing the behavior in the subclass
- To override a method simply use the same name as that of the super class method
- e.g.

```
class Person {
        printInfo() { …. }
}

class Student extends Person {
        printInfo() { …. }
}
```

Equality

- Equality
  - Should be avoided
- Identity equality
  - Type safe
  - Convenient and concise
  - +0 is equal to -0
- Object.is()
  - Type safe
  - Verbose
  - +0 is not equal to -0

Advanced Functions

const num1 = 100

0x2000

100

num1

const num2 = num1

0x3000

~~100~~ 500

num2

num = 500

```
function function2() {
  console.log('inside function2')
}

function2()
```

0x10000

0x20000

function 2 reference

0x 20000

console.log(" ..... ")

object of type function

0x30000

0x 20000

myfunction reference

const myfunction = function2

Function Alias [function alias]

- Another name given to an existing function
- Similar to function pointer in other languages
- E.g.

```
function helloFunction() {
        console.log("inside hello")
}

helloFunction()

const myHelloFunction = helloFunction
myHelloFunction()
```

Arrow functions

- Simpler way to write a function expression
- Why to use them
  - Shorter syntax
  - this derives its value from enclosing lexical scope
- E.g.

```
const greet = () => {
        console.log("welcome to JS")
}

greet()
```

Default Parameters

- Function can assign default values to the parameters
- Such parameters can be optionally passed while calling the function
- E.g.

```
function sayHi(person = "person1") {
        console.log(`hello ${person}`)
}

sayHi()                 // hello person1
sayHi("steve")          // hello steve
```

Variable length arguments function

- A function accepting variable length of arguments
- In JavaScript, every function accepts a hidden parameter arguments which can be used to receive all the arguments
- E.g.

```
function add() {
        let sum = 0
        for (let index = 0; index < arguments.length; index++) {
                sum += arguments[index]
        }
        console.log(`sum = ${sum}`)
}

add(10, 20)             // 30
add(10, 20, 30)         // 60
add(10, 20, 30, 40)     // 100
```

Rest Parameters

- The rest parameter syntax allows us to represent an indefinite number of arguments as an array
- Think about it as the remaining parameters
- If present, the rest parameter must be the last parameter
- E.g.

```
function printInfo(firstName, lastName, ...details) {
        console.log(`firstName: ${firstName}, lastName: ${lastName}`)
        for (let arg of details) {
                console.log(arg)
        }
}

printInfo('steve', 'jobs')
printInfo('steve', 'jobs', 'apple', 'CEO')
```

Spread Operator

- Exactly opposite of rest parameters
- **Spread syntax** allows an iterable such as
  - an array expression or string to be expanded in places where zero or more arguments (for function calls) or elements (for array literals) are expected or
  - an object expression to be expanded in places where zero or more key-value pairs (for object literals) are expected
- E.g.
  - const dateFields = [2020, 3, 25]
  - const d = new Date(...dateFields)

Functional programming

- Functional programming is a programming paradigm, a style of building the structure and elements of computer programs, that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data
- Features
  - Pure functions
  - First class and higher order functions
  - Recursion
  - Type system
  - Referential transparency

programming using function

→ functions are considered as first class citizens

① → functions are treated like variables

② → function can be passed as a parameter to another function

③ → function can be returned as a return value of a function

① declaring function alias ✓

② pass function as a parameter

③ return a function as return value

✓① map

✓② filter

✓③ reduce

✓④ forEach

✓⑤ findIndex