12:30

Page No
Date 1/1/24 YOUVA

Java Script

(ctrl + L Clear consol

Uses
REPL
Read - Evaluate - Print - Loop

Variable:
A variable is simply the name of a storage location

Datatype in Javascript

- Number 5, 5.9
- boolean
- String
- undefined
- null
- Bigint
- Symbol

type of a return number

Numbers in JS
- positive (14) & negative (-9)
- Integer (45, -50)
- floating number with decimal (4.6, -6.9)

Operators
+ , - , * , / , % , 2**3
Power

ex: a + b
operator

NaN in JS
not - A - Number

0/0
NaN-1
NaN * 1
Nan + Nan

Operator precendece
This is the general order of solving an expression

() → ** → *, /, % → + -

let keyword

let age = 23;
age = age + 1

let country;
country = 8.9

const var

var age = 23

var apa = 8.9

Boolen

let isAdult a = true
isAdult b = False

What is Typescript
: static Type, where as is a dynamic typed. Desired by microsoft

null & undefd inJS

undefind : A varion that has no been assind a value

Display message of block

console.log

Truthy & falsy value:
Everything in JS is true or false

This dosen't

string methods (immutable)
actions that can be performed on objects

Format
StringName.method()

str.trim
: Remove space last and fast (not middle)

str.uppercase()
str.tolowercase()
str.indexof("A") - ?/ -1 / not fand

method chaning

str.uppercase.trim();
str.slice(s?, slice(1, 9), str(- num))
replace str.slice("o", "x")
find replace

Array.length
let a = ("a", "b", "c")
a[0] =
a[1]

mixed arr = ["aman", 25, 6.1];

mutable

arry = [1, 2, 3]
arr [10] = 10

arr [1, 2, 3, und, und, und, und ... ,10]

Array methods
push - push at last
pop - last element
unshift - add at start
shift - delete at start
indexof - return index if not -1
includes - true / false
concat - concat 2 string
reverse : reverse();
slice : slice
(start, deleteat, item...item) # splice - remove, replace, add
sort - sort
const (arr) - not overide variable

Array · Refrance
address in memory
== | compare value
=== | compare address
it will contain same path to memory

Get inr
# input
prom(' ' text 'i');
}
for of loop
for (char of "apnacollege"){
console.log(char)

for (char of str)

JS object literals
used to store keyed collection & complex entities

property => (key, values) pair
Objects are a collection of properties

```
JS objects literals = {
    latitude : "28.7041° N"
    longitude : "77.1025° E"
};

Const student = {
    name : Sam,
    age : 18
    m : 77
};
```

Imp
unordered object of key value pair

Thread / twitter post
Create an object like for prop of thread / twitter post which includes:
- username
- content
- like
- reposts
- tags

Get value

```
let student = {
    "name" : "sam",
    mark : 94.4
};

Student ["Name"]   } methods
Student.Name
```

JS automatically converts objects keys to string
Even if we made the number as a key the number will be converted to string

object keys are always be string

```
const obj = {
    1: "a",
    2: "b",
    true: "c",
    null: "d",
    undefined: "e"
}
```

string — values are defined

add / update value:

```
Const studel = {
    name : "Shradha Saini",
    age : 23
    mark : 9x.6
    city : "Pune"
};
```

- Change city to mumbai

```
Student.city = "mumbai"
```

Create

add ⇒ Student.gender = "male"

delete

```
delete student.gender;
```

Nested object

```
Const classInfo = {
    aman : {
        grade : "A+",
        city : "Delhi",
    },
    Saini : {
        grade : "A",
        city : "Pune",
    }
};
```

Array of object

```
Const classInfo = [
    { name: "aman",
      grade: "A+",
      city : "Delhi"
    };
    { Name: "Saini",
      grade : C
      City : Pune
    }
];
```

Math object

Propetis	methos
Math.PI	math.abs(n)
Math.E	math.pow(a,b)
	math.floor(n)
	math.ceil(n)
	math.random()

Console type 'math'

> Math // show all methods and values

Random Integer

```
let num = math.random();
num = num * 10;
num = math.floor(num);
num = num+1
```

OR

```
let random = Math.floor(math.random() * 10) + 1;
```

Not start from 0

Math.random Generate the values like 0.9xxxxxxx

Function in JS

```
function functionName() {
    // do stuff
}
```

Calling :

```
functName();
```

function with Araument. (name, age) {

display name :

```
function printInfo(name, age) {
    console.log(`${name} is ${age} years old`);
}
```

${name}
arg

Return

ret val

Scope :

Scope determines the accessibility of vars objects, and functions from different parts of scope of the code

- Function : func inter curly
- Block { }
- Lexical

Higher order functions

: Takes an or more function as argument

ex

```
function multipleGreet(func, n) {
    for(let i=1; i<n; i++) {
        func();
    }
}
let greet = function() {
    console.log("hello");
}
// call // multipleGreet(greet, 2);
```

Higher order function returns a function

```
function oddEvenTest(request) {
    if (request == "odd") {
        return function(n) {
            console.log(!(n%2 == 0));
        }
    } else if (request == "even") {
        return function(n) {
            console.log(n%2 == 0);
        }
    } else {
        console.log("wrong request");
    }
}
```

method const calc {

```
add : function(a, b) {
    ret a+b;
}
}
```

this keyword in Javascript

```
Const student = {
    name : "sourav",
    age : 23,
    eng : 95,
    math : 93,
    phy : 92,
    getAvg () {
det avg = (this.eng + this.math + this.phy) / 3
    Const.log (avg);
}}
```

arrow function ⇒

Syntax

```
Const func = (arg1, arg2 ..) ⇒ {def}
```

```
Const sum = (a, b) ⇒ { Const.log (a+b)
```

```
Const sum = (a, b) ⇒ (a * b)
```

settimeout (function, timeout)

```
Console.log ("hi there");
Return id = setTimeout ( () ⇒ {
    Console.log ("Apna college");
    } 4000 );
Console.log ("welcome to");
```

set interval (function, timeout) // loop start

```
let id = setInterval ( () ⇒ {
    Console.log ("Apna college");
    } 2000);
```

Stop

```
clearInterval (id);
```

// repetation will be done

Stop in consol command : clearInterval (id)

this keyword with arrow function

arrow	function

Array methods

① arr.forEach (some function or name);
// for of // Reacty und

Ex let arr=[1,2,3,4,5]
// using arrow function
arr.forEach((el) => {
console.log(el);
});
arr.forEach(function (el) {
consol.log(el);
});

map

newArr = arr.map( )

ex
let num [1,2,3,4];
let double = num.map (function (el){
retun el*2;
});

for 'obj'
let apa = studet.map ((el) => {
return marks /10)
});

Filter

ex
let [2, 4, 5, 6, 2, 8, 9];
let even = nums.filter(num) => (num%2 =0)
retun
let (2, 4, 6, 8)

Every

Returns true if every element of array gives true for some function. Else retfy

arry.every ( ); all values consatisfy the condition

ex
① [1,2,3,4].every ((el) => (el %2 ==0));
False
② [2,4].every ((el) => (el %2 ==0));
True
→ Returns Boolean value if condition is true forall

Reduce

Reduce a array to single value
arr.reduce (accumulator, element);

ex [1,2,3,4].reduce ((res, el) => (res+el));
⇒10
function returned value
next element

① maximum in any

let max = arr.reduce ((res, el) =>
({
if (res > el)
res = el
retun res
}

Default param

funcan sum (a, b=3)2
ret a+b
3
sum (2);

V.Imp

Spread : ••• arr

Expands an iteration into multiple values

ex

math.min (1, 2, 3)
math.min (arr[0], arr[1], arr...n)

solve

math.min (••• arr)
// return min from arr

Console.log (••• arr)

let char = [... "hellow"])

with obj literals

```
let data = {
    email: "abc@gmail.com";
    passw : "abcd";
};
let datacopy = {...data, id: 123}
```

+ arr into object

```
let arr = [1,2,3,4,5];
let obj = {...arr};

{ 0 : 1
  1 : 2
  2 : 3
}
```

Rest (opposite of rest)

allows a function to take an indefinite number of arguments and bundle them in an array

Destructuring

Storing value of array into multiple variables

```
let names = ["tony", "ab", "bc", "cd"];
let [ com, win, wa ] = names
```

if ... way

it contains most element

ex

```
const student = {
    name: "Karan",
    age : 14,
    class : 9,
    subject : ["hindi", "english", "md", sa],
    username : "Karan@123",
    password : "abcd",
};

let {username : user, password : secret, city}
= student;
```

Understanding

Page No 2/21 Date 3-12-4

Java Script HTML CSS

DOM

(Document object model)

- This Dom represent a document with a logical tree.
- It allows us to manipulate / change the webpage content (HTML elements)

Selecting Element

get element by id

Returns the elements as an object or null (if not found)

ex

By ID

document. getElementByID (id)

obj

let imgobj = document.getElementById (id)

Console. dir (imgobj);

imgobj. src

imgobj. src = " ";

By class

document . getElementsByClassName ("class na")

HTML collection obj

docum. getElementByClassName ("img"). [0]

[1]

By Tag

document. getElementsByTagName (" ") [0]

docum. getElementByTagName ("p") [1]. innerText = "..."

Query selectors / QuerySelectorAll

Allow use to use any css selector

document. querySelector ("h1"); by tag

document. querySelector ("#description");

document. querySelector (".oldImg");

All

properties and methods

- innerText

Shows the visible text contained in a node

- Textcontent

Shows all the full text

- innerHTML

shows the full markup

ex

let heading = document. querySelector ('h1');

heading. innerHTML = '<u> ${heading.innerHTML} </u>';

manipulating Attributes

obj. getAttribute (attr)

obj. setAttribute (attr, val)

ex: let img = document. querySelector ('img');

img. getAttribute ('id'); // return id

img. setAttribute ('id', 'spidermanImg');

Manipulating style:

```
let img = document.querySelector('img');
for image  console.dir(img)
           img.style.cursor // write all style
```

for h1s
```
let h1s = document.querySelectorAll('h1');
console.dir(h1s)
h1s.style.color = "yellow";
```

Imp → in console log style the only inline css style will be display not all

check object classes

```
obj.classList
classList.add()      // to add new class
classList.remove()   // remove classes
classList.contains() // if class exist
classList.toggle()   // to toggle by 2 try if not add it
```

Navigation

: to navigate on py

```
parentElement
children
previousElementSibling / nextElementSibling
childElementCount
```

adding Element

document.createElement( )

```
document.createElement('p');
let newp = document.createElement('p');
newp.innerHTML = "Hi I am a new p";
```

- appendchild(element)
- append( )
- prepend( ) // at stud append
- insertAdjacentA(whr, element)

appendchild(newp)

```
let body = document.querySelector('body');
body.appendchild(newp);
// add at last
```

append // inset text and as a child

```
let newb = document.createElement('button');
newb.innerHTML = "clickme"
body.c
```

insertadjacent(whr, elem)

beforebegin
afterbegin
beforeend
afterend

- Removechild(elem)
- remove(element)
  b.remove();

Event Listenss

element.addEventListeners Dom events

Events are signals that something has occurd (user inputs / actions

in HTML -> JS

```
<button onclick="console.log('')">click me</button>
```

Dom Events

onclick (when an element is clicked)

onmouseentn (when mouse enters an element)

JS

```
let btn = document.querySelector("button");
console.dir(btn);
btn.onclick = function(){
    alert("button was clicked");
};
```

Event listener

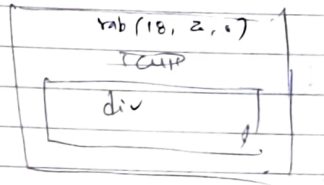
addEventListener (event, callback)

element.addEventListn (event, callback)

event: ~~click~~ drag, Press, dblclick

callback: function

ex btn.addEventListenn ('click", sayHello);
btn.addEventListenn ('click', sayName);




```
let btn = document.querySelector("button");
btn.addEventListener.querySelector("h3");
btn.addEventlistenn("click", function(){
    let h3 = document.querySelector("h3");
    let randomcolor = getRandomcolor();
    h3.innerText = randomcolor;
    let div = document.querySelector("div");
    div.style.background color = randomcolor;
    console.log("color updated");
});
function getRandomColor(){
    let red = math.floor(math.random() * 255);
    let green = math.floor(math.random() * 255);
    let blue = math.floor(math.random() * 255);

    let color = `rgb(${red}, ${green}, ${blue})`;
    return color;
}
```

this in Event Lstnr

when 'this' is used in a callback of event handler or something .it refers to that something

```
btn.addEventListenn("click", function(){
    console.dir(this.innerHTML);
    this.style.backgn = color : Blue;
```

Keyboard Events

Keydown

Keypress

Keyup

```
let inp = document.querySelector("input");
inp.addEventListener("keydown", function(){
    console.log("key was pressed");
});
```

event

code	key
'a'	"a"
'b'	"b"
"semicolon"	";"

```
inp.addEventListener("keydown", function(event){
    console.log(event.key);
    console.log(event.code);
});
```

form event

- submit

```
let form = document.querySelector("form");

form.addEventListener("submit", function(){
    alert("form submitted");
});
```

Prevent default.

```
form.addEventListener("submit", function(evt){
    // evt.preventDefault();
    alert("form submitted");
});
```

Extracting Form data

```
let form = document.querySelector("form");

form.addEventListener("submit", function(event){
    event.preventDefault();
    let inp = document.querySelector("input");
    console.dir(inp);
    console.log(inp.value);
});
```

this.elements

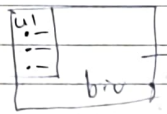
change event : The change event occurs when the value of an element has been changed (only works on <input>, <textarea> and <select> elements)

```
let user = document.querySelector("#user");
user.addEventListener("change", function(){
    console.log("input changed");
    console.log("final value =", this.value);
});
```

Input event:

The input event fires when the value of an <input>, <select>, or <textarea> element has been changed.

Event bubbling



When ul was clicked
div was also be clikkd

*nested event

Stop bubling

```
event.stoppropagation();

ul.addEventListener("click", function(event){
    event.stoppropagation();
    console.log("ul was clicked");
});
```

TODO App


```
let btn = document.querySelector("button"),
let ul = document.querySelector("ul");
let inp = document.querySelector("input");
btn.addEventListener("click", function(){
    let item = document.createElement("li");
    item.innerText = inp.value;
let delBtn = document.createElement("button");
delBtn.innerText = "delete";
delBtn.classList.add("delete");

item.appendChild(delBtn);
ul.appendChild(item);
inp.value = "";
});
let delBtns = document.querySelectorAll(".delete");
for(let delBtn of delBtns){
    delBtn.addEventListener("click", function(){
        let par = this.parentElement;
        console.log(par);
        par.remove();
});}
```

Event Delegation

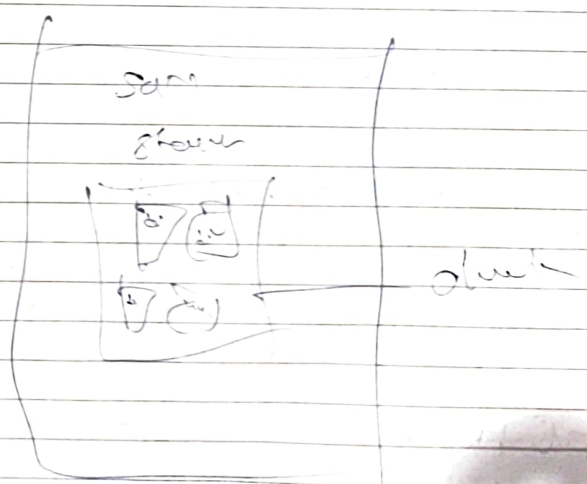
bubbling

Parent → child

event listn ~~cato~~ child

```
ul.addEventListenir("click", function(event){
    if(event.target.nodeName == "button"){
        let listItem = event.target.parentElement;
        listItem.remove();
        console.log("deleted");
});
```

Simon Game



Day

#1 Call stack

#Breakpoints

JS is single Threaded :
one time one thing will be executing

```
setTimeout (() => {
    console.log ("apna college");
}, 2000);
setTimeout (() => {
    console.log ("apna college");
}, 2000);
console.log ("hello...");
```

Output

Hello
apna college

Second callback Hell
Promises :

The promise object represent the eventual completion (or failure) of an asynchronous operation and its resulting value.

Resolve & reject

Day

Revision

```
function saveToDb (data) {
    return new promise ((success, failure) =>
{ let internetspeed = math.floor(math.random()*10)+1;
    if (internetspeed >4) {
        success();
    } else {
        failure();
    }});
}
```

Perry

Async functions

async an await

```
async function greet () {
    return "hellow world"; // returns a promise
}
let Hello= async () => {};  // retun a promise
```

Ex

```
Async function greet(){
    throw "404 page not found"}
    return "hello!";

greet()
  .then ((result) => {
      console.log ("Promise was resolved");
      console.log ("Result was:", result);
    })
  .catch ((err) => {
      console.log ("promise was rejected with err;
      ------------>", err));
    });
```

Accait Keyword

pauses the exection of its surrounding async function until the promise is settled (resolved or rejected)

```
async function show() {
    await colorchange ("violet", 1000);
    await colorchange ("indigo", 1000);
    await colorchange ("green", 1000);
    return "done";
}
```

Pandy

Pandas

Async

await

Handling rejection

Pandy

Page No.
Date 8/11/24
YOUVA

Api : Cappication proaraming Interface)

API
↓
JSON
(data)

Random api

ex : catfact ninja /fact
Dogedapi.com/api/activity
dog.ceo/api/breeds/image/random

JSON
: Javascript object Notation (JSon.org)
or (format of returning of JSON)
(xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<EmployeeData>
  <employee id="34594">
    <firstname>Hello</firstname>
    <lastname>Bank</lastname>
    <HireDate>18/1998</HireDate>
  </employee>
</EmployeeData>
```

JSON

```
{
  "string": "Hi",
  "number": 2.5,
  "boolean": true,
  "null": null
  "object": {"name": "kyle", "age": 24},
  "array": ["Hello", 5, false, null,
  {"key": "value", "number": 6}]
  "array of
```

Paidy

JSON

Accessing Data from JSON

string
- JSON.parse(data) method To parse a string data into a JS object

object
- JSON.stringify(json) method To parse a JS object data into JSON

string
```
let validRes = JSON.parse(data)
console.log(validRes.key)
```

object
```
let [illegible] = JSON.stringify(object) [illegible]
console.log([illegible])
```

API Testing tools

- Hoppscotch
- postman

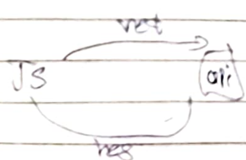
www. Hoppscotch.io

get - get json from
put
post
patch
Delete
Head
Connect

HTTP verbs

Get
Post
Delete

AJAX : Asynchronous Javascript and XML



JSON AJAJ

Parts

Status codes

- 200 ok
- 404 - Not found
- 400 - Bad Request
- 500 - Internal server error

Informational responses (100-199)
Successful Responses (200-299)
Redirection messages (300-399)
Client error responses (400-499)
Server error response (500-599)

add information in URL
Query strings

https://www.google.com/search?q =harry +porter
key value

ex

?name =Saurav & marks =95

HTTP headers

header , values : supply additional information
req , resp.

Get v | [Send] [SAVE]

```
let url = "https://catfact.ninja/fact";
fetch (url)
  .then ((response) =>{
    console.log (response); })
```

Parka fetch()

```
.catch ((err) => {
        Console.log ("Error - ", err);
    });
```

example

```
let url = "https://catfact.ninja/fact";
fetch (url)
    .then ((res) => {
        Console.log (res);
        return res.json();
    })
    .then ((data) => {
        Console.log (data.fact);
    })
    .catch ((err) => {
        Console.log ("ERROR - ", err);
    });
```

with async

```
let url = "https://catfact.ninja/fact";

async function getFacts () {
    try {
        let res = await fetch (url);
        let data = await res.json();
        Console.log (data.fact);
    } catch (e) {
        Console.log ("error - ", e);
    }
    Console.log ("bye");
}
```

Parka

Axios

To create an http request

```
async function getFacts () {
    try {
        let res = await axios.get (url);
        Console.log (res);
    }
    catch (e) {
        Console.log ("Error - ", e);
    }
}
```

fetch CD

How to use

github / axios / installing / CDN

line <script> use in html body

axios.get
axios.post

Advantage

fetch can retur respont not in JSON

AXOS can retun response in JSON

we want to parse

Fetching

Show facts on page

```
let btn = document.querySelector("button");

btn.addEventListener("click", async () => {
    let fact = await getFacts();
    console.log(fact);
    let p = document.querySelector("#result");
    p.innerText = fact;
});

let url = "https://catfact.ninja/fact";
async function getFacts() {

async function getFacts() {
    try {
        let res = await axios.get(url);
        return res.data.fact;
    } catch (e) {
        console.log("error -", e);
        return "No fact found";
    }
}
```

axios

Sending Headers

```
const config = { headers: { Accept: "application/json" } };
let res = await axios.get(url, config);
console.log(res.data);
```

Terminal

The text input and output environment

Git - scm.com

GitBash

Home mac

~ - root directory

git --version - shows the version of git

- ls - - show all commit short list
- clear
- pwd - working directory

Navigation Commands

Inside & outside Directories

- cd - change directory
  cd.. - back to previous dir
  cd../../ - 2 step back

Path

absolute cd /Users/shivam/Desktop

Relative cd Desktop/shivam

/ = root dir → cd /

~ = Home dir cd ~

ls - show which dirs you have

making Directories

mkdir - make directory - folder

Flags

flags are charaters that we pass with commands to modify thair behavir

man ls - give info about ls commands
man mkdir - give info about mkdir commands

with Flags

ls -l - show with properties

ls -a - show all (hidden also)

ls -la - all files with propedire

Touch Command

touch index.html
touch app.js
touch abc.test

Deleting Files & Folders

rm. remove files
rmdir - remy emy folders
rm -rf - removes any folders

recursive force

Git Github

Git
version control system
↓
tools that help to tracks chan in code

what is Github!-
website where we host repositories online

Commit → changes
to final the changes

Using Git

- Command Line git --version
- IDE / code Editors (like vscode)
- Graphical user Interfa (like Gitkraken)

- Command line

git - Shows the all commands
add - add file content to the index
mv - move or rename a file a dirxy
restore - Restore working tree files
rm - Remove files from the working tree and from the index

Examin the history and state

bisect : seach find the commit that introdu a bug
diff : Chanve between commit, comi and worky tuy

log - commit logs
show - various types of oblect
status - working tree status

configuring Git

or [git config --global user.name "myName"
git config --globa user.email "email"

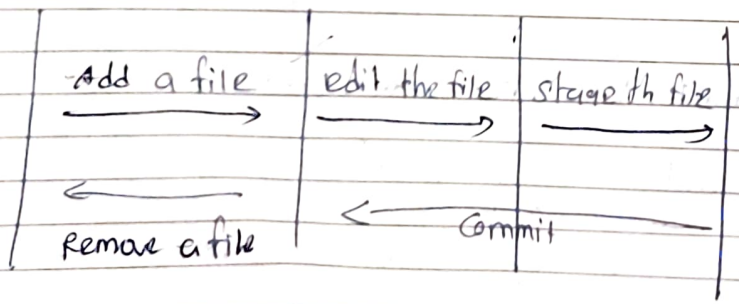
① git config --list
returns user.name = abc
user.email = abcc@email.com

Basic Commands
① • clone - cloning a repository on our local machine
② • status - dispay the state of the code

① git clone <-some link->
② git status

Vscode > terminal > git clone <link>

Files	stats	lifecycle	
(new files)	(constant)	(changed)	
untracked	unmodified	modified	staged



Add & Commit

• add - adds new or changed files in your working directory to the Git staging area
• commit - it r the record of changes
• push - upload local repo contet to remote repo

git add <-file name->
git commit -m "some message"
git push origin main

working → staging → finalstate
dir area commit

ex
git add index.html
git status
git add . // all add
git commit -m "add new file"

git push : vs changed file reflet on git
upload local repo content to remote repo
- git push origin main
branch
push git
fily file

Basic

Basic Commands

- init - used to create a new git repo

```
git remote add origin <-link->
git remote -v (to verify remote)
git branch (to check branch)
git branch -M main (to rename branch)
git push origin main
```

Set origin

```
git push -u origin main (core himself)
git commit -am "add dot"
git push
```

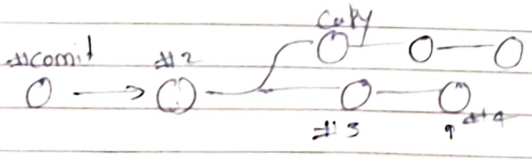
Work Flow

Github
code
Chang
↳
commit

Local Git
code
Chanes
↓
add (stage)
commit (commit chn)
↓
push
PR

Git Branch

Realtn



Branch Commands

```
git branch (to check branch)
git branch -M main (to rename branch)
switch-> git checkout <- branch name -> (to navigate)
new-> git checkout -b <- new branch nam -> (to creat new brch)
git branch -d <- branch name -> (to delete brch)
```

merge branch

```
git push --set-upstrm origin feature
```

Merge code
git diff <- branch name -> (to ~~com~~ comm commit branchs, files & more)
git merge <- branch name -> (to merg + brs

OR

CRet PR

Pull request

It lets you tell others about changes you've pushed to a branch in a repository on Github

---

git pull origin main

---

used to fetch and download content from a remote repo and immediately update to local repo to match that content

Merge conflicts

An event that takes place when Git is unable to automatically resolve differences in code between two commits

Fixing mistakes

Case 1 : Staged changes
git rest <- file name ->
git rest

Case 2 : Commit changes (for one commit)
git reset Head ~1

Case 3 : Commited changes (for many commits)

git reset <- commit hash ->
git reset --hard <- commit hash ->

fork

to clone and make repo of it

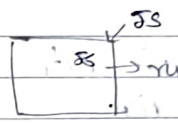
A fork is a new repository that shares code and visibility setting with the original 'upstream' repository.

Fork is reach + copy

very much

Node JS

Javascript runtime environment
It is used for serverside programming
* Node.js not a language, library or framework



Installation
terminal - node
node -v

Node REPL (Read Evaluvate Print loop)

Process : This object provides information about, and control over the current Node.js process
process.argv : returns an array containing the command-line arguments passed when the Node.js process was launched
console.log(process.argv);

Access command line argument
* node script.js Hellow by my Saini

let args: process.args
for (let i=2; i<args.length; i++){
console.log(Hello ${args[i]});
}

start from 2nd place of array

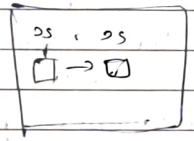
Export in File :

ex b.js content
b.js module.exports = 123 - a special object
a.js const someVar = require("./b"); file b.js
a.js console.log(someVar)

require() - a built in function to include external modules that exist in seprate file.
module.exp - a special object

```
const sum = (a,b) => a+b;
const mul = (a,b) => a*b;
const g = 9.8;
const pi = 3.14;

let obj = {
    sum : sum,
    mul : mul;
    g : g,
    pi : pi }
module.exports = obj;
```

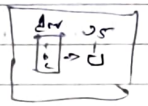


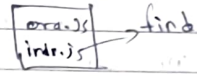
a.js

```
const b = require("./b");
console.log(b.pi);
console.log(b.sum(11,12));
```

module.export → require("path")
obj obj

module.exports
requiring directories

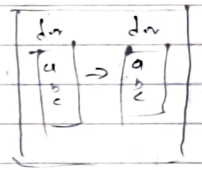


folder from outside


→ find of file

What is npm

NPM (Node package manager)
Nop, npm is standard package manager for Node JS



Installing package . npmjs.com

npm install <- package name ->

ex
figlet

```
const figlet = require("figlet");
console.log(sum);
}
```

Package.json

The package.json file describing and functional metadata about a project such as a name version and dependecies.

npm install
↑
it will install from package.json

npm init,
after creation project

require v/s import

ES6 :- new version of Js

import {sum3} from "./maths.js"

we can't selectively load only the pieces we need with require but with import we can selectively load only the pieces we need which can save many

step 1 : node init

add "export const pi = 3.14"

step 2

package.json

add "type" = "module"

step 3

b.js import {pi} from "./a.js"

Loding is synchronous for "require" but can by asynchronous for "import"

Library — Framework

A library is a collection of pre-written code that can be used to perform specific tasks

A framework is set of pre-written code that provides a structure for developing software application

eg axios — eg express

Express

A Node.js web application framework that helps us to make web applications it is used for server side programming

1) listen incoming request
2) parse
3) route — eg google.com/home/a.b (home = route)
4) suitable response

npm init

npm i Express

```
Const express = require("express");
Const app = express();
```

Ports : the logical endpoint of a network connection that is used to exchange information between a web server and web client.

```
let port = 3000;
app.listen(port, () => {
    console.log("app is listening on port ${port}");
});
```

web localhost:3000

Handling request

```
// app.use((req, res) => {
    console.log("new incoming request");
});
```

HOPPSCOTCH.io

Sending a Response

req / res — http. → textbase

express → convert to obj

```
res.send ("string")  - string
res.send ({
    name : "anu"
    cat : "cat"
})                   - obj
```

-Html res.s.send ("<H1> Hello </H1>");

routes

Routing ex http://localhost:8080/Home/a

is is process of selecting a path for traffic in a network or between or across multiple networks

```
ex  app.get ("/apple", (req,res) => {
        res.send({
            Name : "apple";
            Color : "red",
        });
    });

app.
app.get("*(req, res) => {
    res.send ("this is not res");
});

app.post ("*", (req.res) => {
    res.send ("this is post m*)
});
```

Nodemon

To automatically restart server with code changes

sudo global
~~node npm~~ npm install -g nodemon;

Nodmon index.js

Path Parameters

```
req.params
app.get ('/ig/:username', (req,res) => {
    let {username} = req.params;
    res.send ('this account belong to @${username}');
});
app.get("/ig/:user/:id ?")   // multiple part
```

Query string

req.query http://localhost:3000/search/?q=apple

```
app.get ("/search", (req,res) => {
    let {q} = req.query;
    if (!q) {
        res.send ("No search query");
    }
    res.send ('these are the res for ${q}')
});
```

EJS → ejs.co → make dynamic web pages
Templating

EJS (Embedded Javascript templates)
EJS is a simple templating language that lets you generate HTML markup with plain Javascript.

default set
npm init -y
npm instal ejs

app.set ("view engine", "ejs");

app.get ("/", (req, res) => {
  res.render ("home.js");
});

Views Directory

Const path = require ('path');
app.set ("views", path.join (_dirname, "/views"))

nodemon EJS/aoindex.js → view will find in pond/div
nodemon index.js →
guide error view not found

Interpolation Syntax

Interpolation refers to embedding expressions into marked up text

ejs.co	EJS tag						
<%	<%_	<%=	<%-	<%#	<%%	%>	-%>
Control flow	whitespace	output value	output unescaped	comment	literal <%	end tag	trim whitespace

EJS : HTML → Javascript

Passing data to EJS

app.get ("/rolldice", (req, res) => {
  let num = math.floor (math.random() * 6) + 1;
  res.render ("rolldice.ejs", {diceVal: num});
});

<h1> Your dice give value : <%= diceval %> </h1>

Instagram EJS

Create a basic template for instagram page based on following route

/ig/:username

Conditional statement EJS

dice//

<% if (diceVal == 6) { %>
  <h2> Nice ! Roll dice again. </h2>
<% } %>

Instagram page with EJS

Const instaData = require ("./data.json");
Const {username} = req.params;
let data = instaData[username];
if (data)
{
  res.render ("instagram", {data});
}
else {
  res.send ("not found")
}

Includes

<?- includes ("includes / head.ejs"); ?>

Miscellaneous

// get post request

Get

>Used to GET some response

Data send in query strings
(limited, string data is visible in url)

Post

used to post something
(for create/write/update)

Data send view request body
(any type of data)

Post request body

- Parse post request data

app.use(express.urlencoded({extended: true}));
app.use(express.json()); // for JSON
console.log(req.body); - print body

Javascript OOPS

To structure our code

- prototypes
- new operator
- Constructure
- Classes
- keyword (extends super)

object prototypes:

Prototype are the mechanism by which Javascript objects inherit feature from one another. It is like single template ~~template~~ object that all objects inherit methods and properties from without having their own copy

arr.proto - (refrence)
Array.prototype (actual object)
string prototype

Factory Function

A function that creats object

```
function personMaker(name, age){
    const person = {
        name: name;
        age: age;
        talk() {
            console.log('Hi, my name is ${this.name}');
        }
    return person;
}
const p = personMaker("sam", 20);
const p = personMaker("Lan", 20);
```

New Operater

The new operater lets developers create an instance of a user-defiend object type or one of the built-in object types that has a constructr function

```
function person (name, age) {
    this.name = name;
    this.age = age;
}
person.prototype.talk = function () {
    console.log ('Hi, my name is ' + this.name);
}
let p1 = new person ('sawn', 21);
let p2 = new person ('gww', 20);
```

Classes

classes are a template for creating objects

The constructor method is a special method of a class for creating and initilizing an object instance of that class

```
class Person {
    constructor (name, age)
        this.name = name;
        this.age = age;
    }
    talk () {
        console.log ('Hi my name is ' + this.name);
    }
}
let p1 = new Person ('adam', 28);
```

Rest

Representational State Transfer

Rest is an architectural style that defines a set of constraints to be used for creating web services

CRUD

Create → Read → Update → Delete

CRUD Operation

Get : retrive resource
Post : submit new data to the server
Put : Update existing data
patch : update existing data partially
Delete : removes data

CREATING Restful API

Index	GET	/posts	to get for all post
Create	post	/posts	to add a new post route
View	GET	/posts/:id	to get one post (using id)
Update	PATCH	/posts/:id	to update specific post
Delete	DELETE	/posts/:id	to delete specific post

Redirect

res.redirect (URL)

Create id for posts

UUID package
Universally Unique identifier
npm install uuid

npm - method override (npm method-override

html allows only get - post
method overide - convert the post to patch, Del

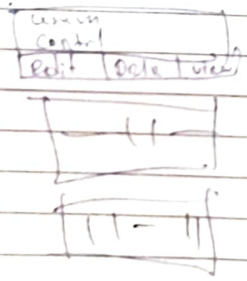
<form method = "post" action: "/resources?_method = DELETE">
<button type = "submit"> DELETE resource </button>
</form>

var express = require("express")
var methodoverride = require('method-override')
var app = express()

app.use(methodOverride('_method'))

Delete

Queries



Database mysql

SQL

Relational Database
eg
mysql, oracle
postgress sql

NoSQL

Non relational Database
eg mongoDb, cassandra
Neo4j

Database

Create database saurav; CREATE DATABASE IF NOT EXISTS bam
drop database saurav; If Exist
use database
Show database

Create Table

Create table saurav(
Colum_name datatype constaint;
column_name datatype constaint,
Colum_name datatype constaint
);

Show Tables => Show all tables

Tables Queries
- create
- Insert
- update
- Alter
- Truncate
- Delete

Constraints
NOT NULL
Unique
DEFAULT
CHECK

Clauses
where
group By
order by

where | having

where is for the table
Having is for a group

Grapphy → having

Grouping is necesry for having

General order

SELECT columns()
FROM tablename
where condition
GroupBy column(s)
Having condition
ORDER by columns(s) Desc Asc

Table Queries

Update (to update existing rows)

```
UPDATE table-name
SET col1 = val1, col2 = val2
where condition;
```

```
SET SQL_SAFE_UPDATES = 0;
// allow / on update
```

- Alter tabua bank
ADD colun autotype varch(50)
- Altr table bank
modify column balance Decimal (12,2);
-

Backend 5 (Node with sql)

Faker npm i @faker-js/faker

To generate fake data
used usernam email password

```
Uet getuser = () => {
    return {
userid faker.datatype.uuid(),
```

mysql2 package

To connect Node with mysql
npm i mysql2

```
Const mysql = requrire ('mysql2');
const conneton = mysql.createConnect({
    host : 'localhost',
    user : 'root',
    databas : 'delta-app'
    passd : "root"
    }
```

USIG SQL in Terminal

```
/usr/local/mysql/bin/mysql -u root -p
Show database;
use database-name;
Show tables;
```

```
Conmectio.query ('show Tables', function (err results, fields)
{ Console.log (results);
  Console.log (fields);
});
  catch (err){
  Console.log (err);
  }
```

Page No
Date

```
CREATE Table User {
    id varchar(50) primary key,
    username varchar (50) unique,
    email varchar (50) unique not null,
    password varchar (50) not null);
```

Insert multiple records

```
let createrandom = () =>
{
    return [
        ];
    };
let data[];
    let q = " Insert into user ( id, username, email, password
    ) values ?
    for
    data.Push

connection.query (q, [data], (err, result) =>
    {
        log (err) , log(result) }
```

Routing

Read

GET / → show no of user in DB
Get / user → show users (email, id, username) etc
Patch / user / :id

Mongodb

Page No
Date 6/2/24

```
sudo systemctl status mongod
sudo systemctl start mongod

sudo apt install mongodb-mongosh
```

cls - clear screen
exit - exit

~~Used~~ Create database

use college.

#BSON Data → Binary JSON

JSON	BSON
UTF 8 string	Binary
string, boolean, Number, Array, object, null	String, boolean, Number (Integer, Float, Long, Decimal 128 ... , Array, null, Date, Bin Data

Document: Mongo stores data in form of document (BSON docs)

Collection: MongoDB stores documents in collections

Collections

```
{ name : "a1",
  age : 18,
  status : 'P'
  groups : [ "Politics", "news" ]
}
```

Insert in DB :-

show collections

NoSQL DB CRUD operation
- Insert Documents
- Query Documents
- Update Documents
- Delete Documents

Insert method

① db.collection.insertOne()
② db.collection.insertMany()

① db.student.insertOne({name: "adam", age: 19, marks: 88})

Show data :- db.student.find()

② db.student.insertMany([{3}, {3}, {3}])

Find in DB

db.collection.find()

db.collection.find({key: value})
db.collection.findOne({key: value})

① Find students where marks>25

db.students.find({marks: {$gt: 25}})

② Find student who live in Delhi or Mumbai

db.student.find({city: {$in: ["Delhi", "mumbai"]}})

③. Find students who scored >25 or live in Delhi

db.studwl.find({$or: [{marks: {$gt: 25}}, {city: "Delhi"}]})

Query Selector

$eq	: equal to
$gt	: greater than
$gte	: greater than equal to
$in	: in
$lt	: less than
$lte	: less than equal to
$ne	: not equal
$nin	: matches none of the values specified in array

Logical

$and	: and
$not	: not
$nor	: nor → both false
$or	: or

Element

$exists	: matchs document that has specified
$type	: selects document if a field is specified type

Evaluation

$expr	: use of aggregation expression with in query
$jsonschma	- validate documents
$mod	: modulo operation
$regex	- values match
$text	- text search
$where	: matches

Array.

Name	Description
$all	: Contains all
$elemmatch	: matches specified condition
$ size	: array fied specified size.

Bitwise

$ bitsAllclear	which set position value : 0
$ bits All set	set position value 1
$ bits Any clear	set value of specif : 0
$. bits Any set	set value of specif 1

Projection Operators

$	array that matchis query condition
$elemmatch	$elemmatch conditon meitch
$ meta	score asign during $text operation
$ slice	limit numbr of elem

Miscellaneous operators

$ comment :	Adds a comment to q : query predicat
$ rand	float between 0 and 1

Update method

db.collection.updateOne()
db.collection.updatemany()
db.collection.replaceOne() - Update all one obj

db.collection.updateOne(<filter>, <update>, <options>)

ex : db.students.updateone({name: "adam"}, {$set: {marks: 99}})

Updates with Aggregation pipeline

$ addFields
$ set
$ Project
$ unset
$ replaceRoot
$ replaceWith

Delete in DB

db.collection.deleteOne(<filter>, <options>)
db.collection.deletemany(<filter>, <options>) or ({}) truncate
db.dropDatabase()

show dbs
db.student.find()

Redund nvm and use node-lv 18.18,

Mongoose

npm i mongoose

```
Const mongoose = require("mongoose")
mongoose.connect("mongodb://127.0.0.1:27017/test");
main().catch(err => console.log(err));

async function main(){
  await mongoose.connect('mongodb://127.0.0.1:27017/test');
}
```

A library that creates a connection between MongoDB & node.js Javascript runtime environment

It is an ODM (Object-Data modeling) library

Schema

: Schema defines the shape of the documents within that collection

```
Const userschema = new mongoose.schema({
  name: string,
  email: string,
  age: Number,
});
```

SchemType:s

String	mixed	uuid
Number	Object id	
Date	Array	
Buffer	Decimal128	
Boolean	Map	

```
Const mongoose = require("mongoose");
async function main(){
  try{
    await mongoose.connect("mongodb://127.0.0.1:27017/test");
  } catch(err){
    throw err;
  }
}

main().then(()=>{
  console.log("connection success");
}).catch(err=>{
  console.error("connection error:",err);
});
```

Schema

```
Const userschema = new mongoose.schema({
  name: string,
  email: string,
  age: Number,
});
```

Models:

Model in mongoose is a class with which we construct documents

```
Const user = mongoose.model("user", userschema);
```

(collection name)

Insert

Inserting one

```
Const user1 = new User({name:'Steve', email:"adu@", age:43});
Const user2 = new User({name:"Eve", email:"eve@a", age 48

user1.save()
user2.save()
```

save() - that will return promises

```
user2.save().then(res) =>{
    console.log(res);
})
.catch((err) =>{ console.log(err)});
```

- loading
like console

Insert multiple

```
User.insertMany([
{ name: "Tony", email: "tony@gmail.com", age: 50},
{ name: "Bruce", email: "bruce@gmail.com", age: 42},
{ name: "Pot", email: "peter@gmail.com", age: 30}
]).then(data) =>{
    Console.log(data);
});
```

Find

```
model.find() // returns a query object (then, null)
```

* mongoose queries are not promises. But they are. then

```
User.find().then((data) =>{
        Console.log(data);
});

User.find({age: {$gte: 42}}).then((data) =>
{
    (console.log(data));
});
```

Update

model.updateOne({name: "Bruce"}, {age: 43}).then

```
((res) => {
    console.log(res);
});
```

model.updateMany({age: {$gt: 45}}, {age: 45}).then

```
((res) => {console.log(res)})
```

Model.findOneAndUpdate()

```
User.findOneAndUpdate({name: "Tony"}, {age: 60},
{new: true}).then(data) =>{
        Console.log(data);
});
```

// it will return the obj and value of the data that will update

Delete

```
User.deleteOne({name:"saurav"}).then((res)=>{
    console.log(res))
});

User.deleteMany({name:"saurav"}).then((res)=>{
    console.log(res)
})
```

Schema

① const userSchema = mongoose.Schema(
```
{
    name: String,
    email: String,
    age: Number,
})
```

② const User = mongoose.model("User", userSchema)); (auto with s)

Schema validations
rules of schema

```
const bookSchema = mongoose.Schema({
    title: {
        type: String,
        required: true,
    },
    author: {
        type: String,
    },
```

Date: 10/2/24

SQL schema → colms → name, datatype, constraints

② mongoose schema types

① required
② default
③ select
④ validate
⑤ get
⑥ set
⑦ alias
⑧ immutable
⑨ transform

= Indexes

① index
② unique
- sparse

```
test: {
    type: String,
    index: true,
    unique: true
```

= String
① lowercase
② uppercase
③ trim
④ match
⑤ enum
⑥ minLength
⑦ maxLength
⑧ populate

= Number	Date
① min	min
② max	max
③ enum	expires
④ populate	

Schema validation

SchemaTypes options

```
Const bookschema = mongoose.schema({
title: {
    type: string,
    required: true,
  },
author: {
    type: string,
  },
price:
    type: Number,
    min: [1, "please enter valid price"],
  },
discount: {
    type: Number,
    default: 0,
  },
genre: [string],
Category: {
    type: string,
    enum: ["Fiction", "non-fiction"],
  },
});
```

Schema validation with Update

if we are using the method `model.findByIDAndUpdate()` it will not validate the validation.

[options.runValidators] <boolean>

```
Books.findByIDAndUpdate(
    "65e1a3a06e",
    {price: -100},
    {runValidators: true}
).then((res) => {
    console.log(res);
}).
```

Custom err

```
.catch((err) => {
    console.log(err.errors.price.properties.messa);
});
```

Mongo with Express

```
Const express = require("express");
const app = express()
Const path = require("path");
Cont mongoose = require("mongoose");

main().then(()=>{
        Console.log("connection successful");
        }).
        Catch((err)=>Consl.log(err));
Async function main(){
    await mongoose.connect("mongodb://127.0.0.1:
    27017/test")
    }
app.set("views", path.join(__dirnm, "views"));
app.set("view engine", "ejs");

    app.get("/", (req,res)=>{
        res.send("working root");
    });
    app.listn(8080, ()=>{
        Console.log("app is lis")
    });
```

Creating the model

chat will have : [id, from, to, meessage, created]

Create chat.js

```
const chat = mongoose.model("chat", chatschmea);
module.exports = chat;
```

↓

inder.js `chonst chat = requw("./chut");`

Initilize Database

Dump data will inserted

```
Vat messar = [ {3}, {3}, {3} ]
chat.insertmany(messages);
```

Using Date

```
<%= chat.created_at.tostring().spiit(" ")[4] %>
<br/>
<%= chat.created_at.tostring().spiit(" ").slice(0,4).
join(" ") %>
```

Major Project

db

- sudo systemctl status mongod
sudo systemctl start mongod.
mongosh

requires - ejs
express
mongodb
node
nodemon

```
app.listen(8080, () => {
    console.log("listening");
});
```

Model listing :

title, descrn, image, price, location, country

Set Default in schema (like image)

1. Create a arrow function
2. and use ternary operator

set : (v) => v === "" ? True : False

Database

show dbs
use Databasename
Show collections
db.listings.find().

delete all : db.listings.deleteMany({});

Date 2/12/24

EJS Mate Npm

ejs-mate

```
const ejsmate = require("ejs-mate");
app.engine("ejs", ejsmate);
```

layout
boilerplate :

Path
views/layout/boilerplate.ejs

```
<div class="container">
    <%- body %>
</div>
```

view ejs

1. Remove all without body tag and add

add :

```
<% layout("/layout/boilerplate") %>
```

public folder for static files

```
app.use(express.static(path.join(__dirname, "/public
```

Navbar

views → includes → navbar.ejs

in

views → layout → boilerplate → body

```
<%- include("../includes/navbar") %>
```

footy? middlewares

express - middleware

It is an intermediary.

Request → [Middleware] → Response

in Express :

Middleware in Express are functions that come into play after the server recives the request and before the response is sent to the client

Common middleware function :

- methodOverride
- bodyParser :
- express static
- express.urlencoded

app.use (express.urlencoded ({ extended : true }));

app.use (express.static (path.join (__dirnam, "/public")))

① middleware can acess req, res oblect
② Chainning ① → ② → ③ passes control
③ Send custom response

what does the middlewares do?

middlewar function can perfom the following task

- Execute any code
- make changes to the request and res object
- End the request - response cycle
- Call the next middleware function in the stack

Our 1st middleware)

```
app.use (() => {
  Console.log ("Hi, I am middleware");
});
```

Using req & res oblect in middleware

```
app.use((req, res) => {
  Console.log ("Hi, I am middlew")
  res.send ("byte");
});
```

app.use (() => { });

if the request can send from server the reqiest direcly come to middleware

middleware has 2 ~~functions~~ steps only

1) responce send
2) next middleware

Using next :

The next middleware function is commonly denoted by a variable named next.

```
app.use ((req, res, next) => {
  console.log( "Time", Date.now())
  next()
})
```

M T W T F S S
Page No.
Date
YOUVA

footer Middlewares

Express - middleware

It is an intermediary:

Request → [Middleware] → Response

in Express :-

Middleware in Express are functions that come into play after the server recives the request and before the response is sent to the client

Common middleware function :

- methodOverride
- bodyParser :
- express.static
- express.urlencoded

app.use(express.urlencoded({ extended : true }));

app.use(express.static(path.join(__dirnam, "/public")))

① middleware can acess req, res object
② Chainning passes control
③ Send custom response

what does the middlewares do?

middleware function can perform the following task :

- Execute any code
- make changes to the request and response objects
- End the request-response cycle
- Call the next middleware function

=> {

req, res) => {

passes the function of middlewa

Token , (req, res) =>{

));

express → guide

Error handler

req, res, next) => {
log(" - - - - ERROR - - - - ");
(err);

err.stack)
send status(500).send("something broke!")

return after lines not be executed

If the current middleware function does not end the request-response cycle, it must call next() to pass control to the next middleware function

Creating Utility Middleware

Logger

```
app.use ((req, res, next) => {
    req.responseTime = new Date (Date.now ()).
toString ();
Console.log ( req.method, req.path, req.responseTime,
    req.hostname);
    next ();
});
```

app.use Callback

- middleware function
- series of middleware
- An array of middleware
- combination of all of the above

API Token as Query string

Let's create an middleware for an api that checks if the access token was passed in the query string or not

passing

middleware → access token

```
app.use ("/root", (req, res, next) => {
    let {token} = req.query;
    if (token === "giveaccess") next();
    else (res.send ("acc
app.get ("/root",
```

passing multiple middleware

```
const checkToken = (req, res, next) => {
    let {token} = req.query;
    if (token === "giveaccess") {
        next();
    }
    res.send ("Access Denied!");
};
app.get ("/api", checkToken, (req, res) => {
    res.send ("data");
})
```

function

passes the function of middleware

```
app.get ("/api", checkToken, (req, res) => {
    res.send ("data");
});
```

Handling Errors → express → guide

- Express Default error handler

```
app.use ((err, req, res, next) => {
    console.log ("----ERROR----");
    next (err);
});
    console.error (err.stack)
    res.send status (500).send ("something broke!")
})
```

Return

→ If return after lines not be executed

Custom Error class

Custom

```
→ class ExpressError extends Error {
    Constructor (status, message) {
        super ();
        this.status = status;
        this.message = message;
    }
}
```

HTTP Response Status codes

1) informational resource (100 - 199)
2) Successful responses (200 - 299)
3) Redirection messages (300 - 399)
4) Client error responses (400 - 499)
5) Server error responses (500 - 599)

Use

```
const ExpressError = require("./ExpressError");
throw new ExpressError (401, "Access DENIED")
```

Default status and message

Error class → custom

```
app.use((err, req, res, next) => {
    let { status = 500, message = "Some Error" } = err;
    res.status (status).send (message);
});
```

Error class

Create an admin route & send an error with 403 status code

Handling Async Error

```
// NEW - show Route
app.get ("chats/:id", async (req, res, next) => {
    let id = req.params;
    let chat = chat.findById (id);
    if (!chat) {
    return next(new ExpressError (404, "chat not found or Deleted"));
    }
    res.render ("edit.ejs", {chat});
});
```

Using try-catch

~~try~~ ~~co~~ app.get ("/path" catch (err) { next (err)

Using wrapAsync

```
function asyncwrap (fn) {
    return function (req, res, next) {
        fn (req, res, next).catch ((err) => next(err));
    }
}

app.get ("/chats/:id", asyncwrap ( async (req, res, next) =>
    { let {id} = req.params;
        res.send ("id", id);
    }));
```

Page No.
Date
YOUVA

Mongoose Error:

```
Const handlevalidationErr => (err) => {
Console.log ("validation error occupied");
 return err;
};
app.use ((err, req, res, next) => {
 Console.log (err.name);
   if (err.name === "validationError"){
     err = handlevalidation Err(err);
   }
   next (err);
});
```

Client side form validation:

1. Client side validation
2. Serverside validation

Bootstrap validation
Input is below form validation
form: class
= needs-validation | novalidate
(while like name)
lable class:
form-lable.
~~fie~~ input class:"
form-control
Validation: message below prompt.
<div class = "valid-feedback"> ok </div>
under the div of field and input

Page No.
Date
YOUVA

Npm Joi

Custom error handler
Custom Wrap Async
 utils → wrapAsync.js
Const wrapAsync = require("./utils/wrapAsync.js");

ExpressError

utils → ExpressError.js

```
Class ExpressError extends Error {
 Constructor (status, message)
 {
  super();
  this.statuscode = statuscode;
  this.message = message;
 }}
module.exports = ExpressError;
```

~~res.sta~~ throw error

throw new ExpressError(303, "listing not found");

Error.ejs → create the error ejs

Imp
Joi.dev api → to validate schema

```
Const Joi = require("joi");
Const listingSchema = Joi.object({
listing : Joi.object({
  title : Joi.~~object~~ ({.string(). required().
  ~~title~~
  description: Joi.string().required(),
```

SQL Relation

Mongo Relationships

SQL (via Foreign Keys)

- one to one
- one to many
- many to many

- one to one - country → president, vice president
  one to many - user - post
  many to many - many student - many subject

- one to many / approach 1 (one to few)
  Store the child document inside parent

```
{
  _id : objectId("6555..."),
  username : 'sherlockholmes',
  address : [
    { location : '221B Baker Street', city : 'London' },
    { location : 'P36 Downtown', city : 'London' }
  ],
  __v : 1
}
```

child document

- one to few Implementation

```
const mongoose = require("mongoose");
main().then(() => console.log("connection successful"))
.catch((err) => console.log(err));
const { schema } = mongoose;
async function main() {
  await mongoose.connect("mongodb://127.0.0.1:27017/relationDemo");
```

```
const userSchema = new schema({
  username : string,
  address : [
    {
      location : string,
      city : string,
    }],});

const User = mongoose.model("User", userSchema);

const addUsers = async() => {
  let user1 = new User({
    username : "sameer",
    address :
      [ location: "221B Baker Street",
        city : "London"
      ]
  });
user1.addresses.push({location : "P32 Street", city : "London"});

await user1.save();
}
addUsers();
```

Mongo Relationship

one to many

Store a reference to child document inside parent

Parent -> child

// one to many relationship in mongod

```
const mongoose = require("mongoose");
const {schema} = mongoose;

main().then(() => console.log("db connected"))
.catch((err) => console.log(err));
async function main(){
    await mongoose.connect("mongodb://127.0.0.1:27017/relation");
}
```

order schema

```
const orderschema = new schema({
    item: string,
    price: number,
});
```

customer schema

```
const customerschema = new schema({
    name: string,
    orders: [
        {
            type: schema.Type.objectID,
            ref: "order",
        },
    ],
});
```

cart ref

ref

```
const order = mongoose.model("order", orderschema);
const customer = mongoose.model("customer", customerschema);

const addOrder = async() => {
    let res = await order.insertMany([
        {
            item: "ball",
            price: 100,
        },
        {
            item: "chips",
            price: 10,
        },
        {
            item: "chocolate",
            price: 10,
        }
    ]);
    console.log(res);
    return res;
};

const addcustomer = async() => {
    let cust1 = new customer({
        name: "saumya",
    });
    let order1 = await order.findone({item: "chips"});
    let order2 = await order.findone({item: "ball"});

    cust1.orders.push(order1);
    cust1.order.push(order2);
    let result = await cust1.save();
    console.log(result);
}
// call function addorder();
    addcustomer();

const display = async() => {
    let records = await custom.find({});
    console.log(records);};

call // display();
```

Mongo Relationship

Populate : is the process of automatically replacing the specified path in the document with documents from other collection

```
Const findCustoms : asunc () => {
    Let result = await custorr.find ({}).
• populate ("orders");

    Console.log (result); // for oblet
})          result [0] // for full info

// it will retn the custorr. order in brif
   it will not retun the oblet id it will
   retn full oblet of it
```

Child > Parent.

One to many (Approach 3 (com to squillions.)

the ~~post~~ user has mutiple post that the upproach of post maks a user

∴ which post posted by user it will reduce th

```
Const : userSchema : new schema ({
    username : string,
    email : string,
  });
Const postschema = new schema ({
    Content : string,
    likes : Number,
```

User

```
Const userschema = new schema ({
    username : string,
    email : string,
});
Const postschema = new schema ({
    Content : string,
    likes : Number,
    user :
    type : schema.Types.objectId,
    ref : "user",
}, {});
```

Imp

```
Const User = mongoose.model ("user", userSchema);
Const pst = mongoose.model ('pst', postSchema
Const getData = asunc () => {
    Let result = await Post.findOne ({}).populate
("user", "username");
    Console.log (result);
});
getData ();
```

populate retn the speay oblet Data in the find one retns

wandw

Handling Deletion
using : Mongoose middleware

We can use 2 middlewares:
- Pre - run before the query is executed
- Post - run after the query is executed

```
customerSchema.post("findOneAndDelete", async(customer)
{
  if(customer.orders.length){
    let res = await Order.deleteMany({_id:{
    $in: customer.orders}});
    console.log(res);
    }});
```

make before schema define

New model Review

- Comment
- Rating (1 to 5)
- Created At

Remove the one to many relationship object from the child schema

ex :- listing → review → after deleting a review the inner of listing review object will remain.

$ Pull