Day -1

Ajax (Asynchronous JavaScript and XML)

-> It is a **Combination of Xml & JS**.
-> It is used to increase the **speed** of web pages.
-> It can **reflect a particular part of a web page**. You don't need to reflect or **reload the whole web page**.

Ajax Uses the process states:
0 **Uninitialized state**
1 **Loading state**
2 **Loaded state**
3 **Processing state**
4 **Complete state**

Ajax uses following type of request status:
200 - request is **Complete** & have **good status**
300 - **Bad status**

```
< Script >
function show () {
    Var obj = new **XMLHttpRequest** (); // class in browser (predefined)
    if (obj) { // object is initialized?
        obj.open ("GET", "first.php"); // open (sending req to the server to
execute .php file)
// Server response to client property of object
ob.onreadystatechange = function ()
{ // return current state
    Console.log (ob.readyState);
    if (ob.readyState == 4 && ob.status == 200)
    {
        P.innerHTML = ob.responseText;
    }
}
ob.send (null); // Breaking connection with server
}
</ Script >
```

```
< div id = "P" > </ div >
< input type = "button" value = "Call by ajax" onclick = "Show()" >

// .php file
< h1 > Welcome to server 2023 < / h1 >
<? php
echo "welcome to track the information ";
echo "< br > here is same more information";
?>

http: // Localhost /ajax1 / file_name.html
```

Day- 2

MySQL

-> To Create database:
mysql > create database database_name ;

-> To open db:
mysql > use database_name ;

-> To Create table:
mysql > create table table_name (id int(10), name varchar(30), Salary decimal (12,2));

-> To Insert record:
mysql > insert into table_name values (100, 'Happy', 9000);

-> To See all records:
mysql > Select * from table_name ;

-> To see only name:
mysql > Select name from table_name ;

mysql > Select name, Salary from t_name ;

-> To records above something:
mysql > Select * from t_name where Salary > 15000 ;

-> To in between:
mysql > Select * from t_name where Salary > 12000 and Salary < 15000 ;

-> To Modify particular record:
mysql > update table_name Set name = 'old_name' where id = 102 ;
mysql > update t_name Set name = 'new_name', Salary = 13000 where id = 111 ;

-> To delete:
mysql > delete from t_name where id = 101 ;

-> To See structure:
   mysql > desc t_name ;

-> To add attribute to table:
   mysql > alter t_name add designation Varchar(30) ;

-> To Set designation of all:
   mysql > update t_name set designation = 'clerk' ;
   mysql > update t_name Set designation = 'manager' where id in (101, 102, 103) ;

 # Group functions

-> Sum of all Salary:
   mysql > Select sum(Salary) from t_name ;

-> Highest salary:
   mysql > Select max(Salary) from t_name ;

-> Min Salary:
   mysql > Select min(Salary) from t_name ;

avg, count are also group functions.

Day-3

Node.js & Modules

Download
Node Js is a server side library & is
used to create the server & Used as a
backend.
+ Make folder
+ Create file main.js
```
  console.log('welcome to server');
```
Cmd node main.js

 # Modules
```
exports.myDateTime = function() {
  return Date();
}

exports.CubeServer = function(req, res) {
  res.writeHead(200, {'Content-type': 'text/html'});
  res.write('The date and time currently are: <div>' + dt.myDateTime() +
'</div>');
  res.end();
}

res.end();
listen (8080);

# Module 2a.js
exports.Cube = function (n) {
  return n * n * n;
}
exports.Factorial = function (n) {
  var r = 1;
  while (n > 0) {
    r = r * n;
```

```
    n--;
  }
  return r;
}
exports.Power = function (n, p) {
  var r = 1;
  while (p > 0) {
    r = r * n;
    p--;
  }
  return r;
}
```

Day- 4

 # Module 2b.js

```
var http = require('http');
var dt = require('./module 2a.js');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-type': 'text/html'});
  res.write('<h3>Cube: </h3>' + dt.Cube(3));
  res.write('<h3>Factorial: </h3>' + dt.Factorial(5));
  res.write('<h3>Power: </h3>' + dt.Power(2, 3));
  res.end();
}).listen(8080);
```

 # Module 3.js

```
var http = require('http');
var url = require('url');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-type': 'text/html'});
  var q = url.parse(req.url, true).query;
  var txt = q.id + ' ' + q.name;
  console.log(txt);
  res.end(txt);
}).listen(8080);
```

// Reading Values from URL posted as query string

File 3.html

```
<form action = "http://localhost:8080/" method="get">
  Name <input type = "text" name = "name" id="name">
  <input type = "submit">
</form>
```

File 4.html

```
<form action = "http://localhost:8080/">
```

```
  Enter a no <input type = "text" name = "n" id="n">
  <input type = "submit">
</form>
```

File 4.js

```
var http = require('http');
var url = require('url');
http.createServer(function (req, res) {
  var q = url.parse(req.url, true).query;
  var t = parseInt(q.n);
  res.write('Square = ' + (t * t) + 'Cube = ' + (t * t * t));
  res.end();
}).listen(8080);
```

Day- 5

File System and Routing (Express/Handlebars)

 # File System

```
var fs = require('fs');
fs.appendFile('mydemo.txt', 'Hello content for system', function (err) {
  if (err) throw err;
  console.log('Saved!');
});
```

```
# Routes in node is using Express & HBS
// Folder structure:
// root/
//   views/
//     home.hbs
//     services.hbs
//     about.hbs
//   public/
//   index.js
```

```
# Index.js in FS + HBS
const path = require('path');
const express = require('express');
const bodyparser = require('body-parser');
const app = express();
const hbs = require('hbs');

app.set('view engine', 'hbs');
app.set('views', path.join(__dirname, 'views'));

// Node Package Manager (NPM)
console.log('my path = ');
console.log(__dirname);
// npm install express hbs body-parser
```

```
// route for home
app.get('/', (req, res) => {
  console.log('Welcome to home page');
  res.render("Service");
});

// route for services
app.get('/services', (req, res) => {
  console.log('Services');
  res.render("Services");
});

// route for about
app.get('/about', (req, res) => {
  console.log('Welcome to about page');
  res.render("about");
});

// Server listening
app.listen(8080, () => {
  console.log('Server is running at port 8080');
});
```

Day- 6

MySQL Connection and Queries

```
var mysql = require('mysql');
var con = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'user_password', // placeholder
  database: 'oopsall'
});

con.connect(function(err) {
  if (err) throw err;
  console.log('connected');
});

var sql = "CREATE TABLE Customers (name VARCHAR(25), address
VARCHAR(200))";
con.query(sql, function(err, result) {
  if (err) throw err;
  console.log("table created");
});

// Install mysql -> npm install mysql

// Add record
var sql = "insert into Customers values ('admin', '45')";
con.query(sql, function(err, result) {
  if (err) throw err;
  console.log('record inserted');
});

// Update record
```

```
var sql = "update Customers set address = 'abc demo' where name =
'admin'";

// Select record
var sql = "Select * from Customers";

// Delete record
var sql = "delete from Customers where name = 'admin'";
```

Day- 7

HBS + MySQL CRUD Implementation

```
// HBS + MySQL CRUD
# -> View
// file: index.hbs
<form action = "/saveproduct" method = "Post">
  Product Name <input type = "text" name = "name">
  Product Price <input type = "text" name = "price">
  <input type = "submit" value="Submit">
</form>

# Product List
<link href = "bootstrap.css" rel = "stylesheet"> // likely referring to a CDN
link
<a href = "http://localhost:8080/product/add">Add New Product</a>

<table class="table table-hover">
  <thead>
    <tr class="warning">
      <th>Product ID</th>
      <th>Product Name</th>
      <th>Price</th>
      <th>Action</th>
    </tr>
  </thead>
  <tbody>
    {{#each results}}
    <tr class="info">
      <td>{{id}}</td>
      <td>{{Product_name}}</td>
      <td>{{Product_Price}}</td>
```

```
      <td>
        <a href="http://127.0.0.1:8080/product/delete/{{Product_id}}"
class="btn btn-danger">Delete</a>
        <a href="#" class="btn btn-success">Edit</a>
      </td>
    </tr>
    {{/each}}
  </tbody>
</table>

# Edit form
<form action = "/update" method = "Post">
  {{#each results}}
  Product Name <input type = "text" name = "name" value =
"{{Product_name}}">
  Product Price <input type = "text" name = "price" value =
"{{Product_Price}}">
  <input type = "hidden" name = "id" value = "{{Product_id}}">
  <input type = "submit">
  {{/each}}
</form>

# More in index.js // referring to the server-side logic for the CRUD
operations
```

Day- 8

SQL Joins and Set Operations

Tables (Used for Joins):

// Table: employee

id	name	Salary	deptno	mgrno
100	admin	7000	10	104
101	SS	5000	15	102
102	abs	6000	11	103
103	as	7000	10	14
104	hr	8000	12	0

// Table: dept

deptno	dname	loc
10	ac	chd
11	sales	mohali
12	bc	pkl
15	cc	abc

Inner Join :

Syntax -> Select * from employee, dept where
employee.deptno = dept.deptno;

OR

Select * from employee E, dept D where
E.deptno = D.deptno;

// Result Table structure showing matched rows (id 100, 102, 103, and 101 if 15 is matched)

```
id | name  | Salary | deptno | deptno | dname | loc
-----------------------------------------------------
100| admin | 7000   | 10     | 10     | ac    | chd
102| abs   | 6000   | 11     | 11     | sales | mohali
103| as    | 5000   | 10     | 10     | ac    | chd
```

```
Select name, D.deptno from employee E, dept D
where E.deptno = D.deptno;
```

// Result Table showing matched name and deptno

```
name | deptno
------------
Admin  | 10
abs    | 11
as     | 10
```

Day-9

 # Left Join :
Same tables
Select A.name, B.dname from employee
as A left join dept as B on
A.deptno = B.deptno;

// Result Table showing all left records (employee) and matched right records (dept)
name | dname
---
admin| ac
SS | cc
abs | sales
as | ac
hr | bc
// It takes left all & right which matches.

Select * from employee as A left join dept as B on
A.deptno = B.deptno;

Right Join :
Select A.name, B.dname from employee as
A right join dept as B on A.deptno = B.deptno;

// Result Table showing all right records (dept) and matched left records (employee)
name | dname

Admin | ac
SS | cc
abs | sales
as | ac

NULL | abc // Assuming 'abc' is another department
// Get right all and left that matched, what did not matched fills with null.

Full Join :
Full -> left query union right query
// Result Table gets both values left & right
name | dname
---
admin| ac
SS | cc
abs | sales
as | ac
hr | bc
NULL | abc // Assuming 'abc' is another department
NULL | NULL // Not clear from table structure

Day- 10

 # Set Operations

// Table: Cities
name
---
chd
mohali
pkl
delhi
punjab

// Table: branch
dname
---
rajasthan
chd
punjab
pune
himachal

Minus (Set Difference)
Minus -> Values that are in 1st but not in 2nd
Select * from Cities where name not
in (Select * from branch);

// Result Table showing names from Cities not in branch
name
---
mohali
pkl
Delhi

Union // (Written near the end of the Minus query)

Intersect (Set Intersection)
Intersect : Common in both
Select * from Cities where name in (Select * from
branch);

// Result Table showing names common to both
name
---
chd
punjab

Self Join

// Table: employee (Modified/Different structure shown for Self Join)
id | name | Salary | deptno | mgrno
---
100 | admin| 4000 | 10 | 104
101 | SS | 5000 | 15 | 102
102 | abs | 6000 | 11 | 103
103 | as | 7000 | 10 | 14
104 | hr | 8000 | 12 | 0

Select * from employee E, employee M where
E.deptno = M.mgrno; // The join condition seems to be E.mgrno = M.id for reporting structure

// Result Table structure showing self-join result based on E.deptno = M.mgrno

```
id | name | Salary | deptno | mgrno | id   | name   | Salary
-------------------------------------------------------
102 | abs   | 6000    | 11       | 103    | 103  | as        | 7000
103 | as     | 7000    | 10       | 14      | 104  | hr        | 8000
104 | hr      | 8000    | 12       | 0       | 100  | admin  | 4000
```

Select E.name, M.name from employee E, employee M where E.mgrno = M.id;

// Result Table showing employee name and their manager's name

```
name | name (Manager)
--------------------
abs        | as
as         | hr
Admin    | hr
```

Day- 11

SQL - Database Setup and Joins

1) Create db
2) Create table employee (id int(10) Primary Key auto-increment, name varchar(20), salary decimal (7, 2), deptno int(20));

3) Create table department (id int(10) Primary Key auto-increment, dname varchar(20), loc varchar(20));

4) Create table project (Pid int(10), eid int(10), Pname varchar(20));

5) Insert record into department:

id	dname	loc
10	ac	chd
11	sales	mohali
12	marketing	pkl

6) Insert into employee

id	name	Salary	deptno
100	admin	7000	10
101	SS	8000	15
102	abs	9000	11
103	as	5000	12
104	F	13000	10
105	E	15000	11

7) Insert into Project

Pid	eid	Pname
200	100	ACC
201	105	MCC
202	102	BCC
203	101	BCC
204	104	ACC
205	105	BCC

8) Select * from employee, dept where employee.
deptno = department.id;

9) join three tables: where emp.deptno =
dept.id & emp.id = Project.eid

10) Select * from ... Make alias ^
get -> e.id, e.name, pro.pname
get -> e.name, salary, deptno, dname, loc,
pname

// Additional fragments

```
app.get('/demo', function(req, res)
  // json parser online fr
  // write query
// # dummy JSON
// Product make Jabaray in json
// let 100 employees api feth data &
// put limit on records (4 to 10)
```

 # Regular Expression

^ -> Starts with

$ -> Ends with

[] -> Specific character

. -> Instance of ""

| -> OR operator

. {n} NUMBER & $ -> Specific Length of String

Day- 12

React Setup and Components

 # React

folder -> react_april2023

-> In cmd npx create-react-app one

-> go to folder cd one

-> cmd npm start

// open file App.js

// Src -> App.js

 # App.js

```
function App() {
  return (
    <div>
      <h1> Welcome to first react app </h1>
    </div>
  );
}
export default App;
```

-> Src -> Component

Code -> open in VS Code.

```
// If New file Second.js
function Second() {
  return (
    <div>
      <div> Welcome to 2nd react app </div>
    </div>
  );
}
export default Second;
// In index.js
import App from './App';
import Second from './Second';

// To get App.js in Second.js
import App from './App';
function Second() {
  return (
    <div>
      <App />
      <h1> 2nd App </h1>
    </div>
  );
}
```

```
export default Second;


# Make new folder 'Components'
// Files: Header.js, Footer.js, Content.js


# Header.js
function Header() {
  return (
    <div>
      <h1> This is Header </h1>
    </div>
  );
}
export default Header;
```

Day- 13

Footer.js

// Same function structure

Content.js

// Same function structure

// In Second.js

```
import App from './App';
import Header from './Components/Header';
import Footer from './Components/Footer';
import Content from './Components/Content';
function Second() {
  return (
    <div>
      <Header />
      <Footer />
      <Content />
    </div>
  );
}
export default Second;
```

Day-14

Hooks : Function Components

```
import { useState } from "react";

function MyStateApp () {

  const [color, SetColor] = useState('red');

  function ChangeColor(e) {

    SetColor(e.target.value);

  }

return (

  <>

    <h1> My fav Color is {Color} </h1>

    <input type="text" value="{Color}" onChange={handleChange} />

    <button type="button" onClick={() => SetColor("Blue")}> Value =
"Blue" </button>

    <button type="button" onClick={() => SetColor("Red")}> Value = "Red"
</button>

  </>

);

export default MyState4;
```

```
// Component 2: Changing State on button click

import React, { useState } from "react";
```

```
function MyState2() {
  const [Color, SetColor] = useState("red");
  function Show1() {
    SetColor("blue");
  }
  function Show2() {
    SetColor("red");
  }
  return (
    <>
      <h1> My fav Color is {Color} </h1>
      <button type="button" onClick={Show1}> Blue </button>
      <button type="button" onClick={Show2}> Red </button>
    </>
  );
}
export default MyState2;
```

Day- 15

// Component 3: Handling multiple states and input change

```
import React, { useState } from "react";
export default function MyBank() {
  const [AccNo, SetAccNo] = useState("");
  const [Name, SetName] = useState("");
  const [Balance, SetBalance] = useState("");

  function Show1(e) {
    SetAccNo(e.target.value);
  }
  function Show2(e) {
    SetName(e.target.value);
  }
  function Show3(e) {
    SetBalance(e.target.value);
  }
  function ShowMe() {
    console.log(AccNo);
    console.log(Name);
    console.log(Balance);
```

```
    document.getElementById("h").innerHTML =
      "Acc No = " + AccNo + "Name = " + Name +
      "Balance = " + Balance;
  }

  return (
    <>
      <h1> Bank details </h1>
      AccNo <input type="text" name="t1" value={AccNo}
onChange={Show1} />
      Name <input type="text" name="t2" value={Name}
onChange={Show2} />
      Balance <input type="text" name="t3" value={Balance}
onChange={Show3} />
      <input type="button" value="Submit" onClick={ShowMe} />
      <div id="h"></div>
    </>
  );
}
// MyState8.js
// import { useState, createContext, useContext } from "react";
```

Day- 16

React Routing and Component Structure

 # Routes + bootstrap

```
// How to create routes & navigate to different pages in
// React
// npm install create-react-app first
// npm install bootstrap
// npm install react-bootstrap
// npm install react-router-dom  // in first folder

// Components {folder}
// go to src -> Components
//   L login
//   L register
//   L content
//   L about

function About() {
  return (
    <div>
      <h1> This is about </h1>
```

```
    </div>
  );
}
export default About;

// Same with others.

// In App.js
import Contact from './Components/Contact'
// import Login from './Components/Login'
// import Register from './Components/Register'

function App() {
  return (
    <div>
      <h1> Hello </h1>
    </div>
  );
}

function App() {
  return (
    <div>
```

```
      <h1> Welcome to routes </h1>

      <Routes>

        <Route path="/contact" element={<Contact />} />

        <Route path="/admin" element={<Contact />} /> // Example route
path

      </Routes>

    </div>

  );

}

// import { ... } in Src -> index
```

Day- 17

React List Rendering and use effect

 # List

```
import React from 'react';
import ReactDOM from 'react-dom';

function App() {
  const myList = ['Peter', 'Sachs', 'Kevn', 'Dhoni', 'Aksh'];

  // List rendering with li
  const listItems = myList.map((demo) => {
    return <li> {demo} </li>
  });

  // List rendering with td, tr, b (for table)
  const listItems2 = myList.map((demo) => {
    return <tr><td><b>{demo}</b></td></tr>
  });

  return (
    <>
      <h1> {myList} </h1>
      <ul> {listItems} </ul>
      <table border="1">
        <tbody>
          {listItems2}
        </tbody>
      </table>
    </>
  );
}
export default App;
```

Day- 18

 # List 2

```
# import { useState, useEffect } from "react";
// import ReactDOM from "react-dom";

const Home = () => {
  const [Data, SetData] = useState(null);

  // Fired after every render
  useEffect(() => {
    fetch("https://jsonplaceholder.typicode.com/users") // Sample API
      .then((res) => res.json())
      .then((data) => SetData(data));
  }, []); // [] means effect runs only once after initial render

  return (
    <>
      <table border="1" width="80%">
        <thead>
          <tr>
            <th> Name </th>
            <th> Username </th>
            <th> Email ID </th>
          </tr>
        </thead>
        <tbody>
          {Data && Data.map((item) => {
            return (
              <tr key={item.id}>
                <td> {item.name} </td>
                <td> {item.username} </td>
                <td> {item.email} </td>
                <td> {item.address.city} </td> // Assuming city is nested in
address
              </tr>
```

```
            );
          })}
        </tbody>
      </table>
    </>
  );
};
export default Home;
```