Node JS

- **Node JS.**
- Node JS is a Java Script runtime built on Chrome's V8 JavaScript engine.
- Node JS is a JavaScript running on the server.
- **History of Node JS.**
- In 2009 an idea came in the mind of Ryan Dahl (Google Engineer) that why not run JavaScript outside browser, so he took V8 engine and embedded in a C++ program and called it Node.exe later on became NodeJS.
- **When to use NodeJS.**
- I/O bound.
- Data streaming application .
- Data streaming application such as Netflix, Instagram etc.
- Real time chat application.
- Real time chat application such as WhatsApp etc.
- **Installation Node JS.**
- **Installation VScode.**
- **Node JS REPL.**
- The REPL feature of Node is very useful in experimenting with Node.js codes and to debug JavaScript codes.
- Read = Read's user's input, parses the input into JavaScript data-structure, and stores in memory.
- Eval = Takes and evaluates the data structure.
- Print = Prints the result.
- Loop = Loops the above command until the user presses ctrl-c twice.
- It is called command prompt.
- **Core modules file system.**
- Consider modules to be the same as JavaScript libraries. A set of functions you want to include in your application.
- Node.js has a set of built-in modules which you can without any further installation.

- **Code Node.js.**
- //files system require all function of files system.
- const fs = require("fs");
- //creating a new file.
- // fs.writeFileSync("read.txt","Welcome to my first file");
- // fs.writeFileSync("read.txt","Welcome to my first Shaan file");
- // fs.appendFileSync("read.txt", " i am fine and How are you!");
- const buf_data = fs.readFileSync("read.txt");
- console.log(buf_data);
- //data convert in string.
- org_data = buf_data.toString();
- console.log(org_data);
- //file name rename.
- fs.renameSync("read.txt", "readwrite.txt");
- Node.js includes an additional data type called Buffer.
- Not available in browser’s JavaScript.
- Buffer is mainly used to store binary data.
- While reading from a file or receiving packets over the network.
- **NodeJS FS CRUD Operation.**
- **create a folder name it Shaan.**
- **create a file in it name bio.txt and data into it.**
- const fs = require("fs");
- fs.writeFileSync("bio.txt","Hello my name is Shaan");
- **add more data into the file at the end of the existing data.**
- fs.appendFileSync("bio.txt"," Saifi");
- **read the data without getting the buffer data at first.**
- const data = fs.readFileSync("bio.txt","utf8");
- console.log(data);
- **rename the file name to mybio.txt.**
- fs.renameSync("bio.txt","mybio.txt");
- **now delete bot the file and the folder.**
- fs.unlinkSync("mybio.txt");

- **NodeJS File System in Asynchronous.**
- **Asynchronous:** synchronicity means that if JavaScript has to wait for an operation to complete, it will execute the rest of the code while waiting.
- const fs = require('fs');
- fs.writeFile("read.txt","Hello my name is Shaan",(err)=>{
-     console.log("file is created");
- });
- fs.appendFile("read.txt", " Saifi",(err)=>{
-     console.log("add");
- });
- fs.readFile("read.txt","utf8",(err,data)=>{
-     console.log(data);
- });
- We pass them a function as an argument a callback that gets called when that task completes. The callback has an argument that tells you whether the operation completed successfully. Now we need to say what do we when fs.writefile has completed (even if it’s noting), and start checking for errors.
- **Node.js Operating System modules.**
- **Docs in NodeJS.**
- The operating system module provides operating system-related utility methods and properties. It can be accessed using.
- **Code in NodeJS.**
- const os = require('os');
- // check the architecture use 32bit and 64bit.
- console.log(os.arch());
- console.log(os.hostname());
- console.log(os.platform());
- console.log(os.tmpdir());
- console.log(os.type());
- //Returns the amount of free system memory in bytes as an integer.
- const freeMemory = os.freemem();
- //convert the bit into gigabite.
- console.log(`${freeMemory/1024/1024/1024}`);
- const totalMemory = os.totalmem();
- console.log(`${totalMemory/1024/1024/1024}`);

- **Path modules in NodeJS.**
- The node:path module provides utilities for working with file and directory paths. It can be accessed using:
- const path = require('path');
- console.log(path.dirname('D:/software/DATA/NodeJS/Path Modules/Path.js'));
- console.log(path.extname('D:/software/DATA/NodeJS/Path Modules/Path.js'));
- console.log(path.basename('D:/software/DATA/NodeJS/Path Modules/Path.js'));
- const mypath = path.parse('D:/software/DATA/NodeJS/Path Modules/Path.js');
- console.log(mypath.name);
- console.log(mypath.root);
- **How to CREATE and EXPORT Our Own Modules in Node JS.**
- **Open.js**
- const add = (a, b) => {
-     return a + b;
- };
- const sub = (a, b) => {
-     return a - b;
- };
- const mult = (a, b) => {
-     return a * b;
- };
- const div = (a, b) => {
-     return a / b;
- };
- module.exports = {add, sub, mult, div};
- **index.js**
- const {add, sub, mult, div} = require('./open');
- console.log(add(5, 5));
- console.log(sub(10, 5));
- console.log(mult(10, 5));
- console.log(div(10, 5));

- **how to import NPM in Nodejs.**
- Npm is a package manager for the JavaScript programming language. It is the default package manager for the JavaScript runtime environment Node.js.
- **Installation this NPM.**
- **Npm init.**
- (description:Shanelhai node tuts. author: Shanelhai.)
- **npm i chalk.**
- **Operation in chalk.**
- const chalk = require('chalk');
- console.log(chalk.blue('Hello world!'));
- console.log(chalk.blue.italic('Hello world!'));
- console.log(chalk.blue.italic.underline.inverse('false'));
- **npm i validator.**
- const chalk = require('chalk');
- const validator = require('validator');
- console.log(chalk.blue.italic.underline.inverse('false'));
- const res = validator.isEmail('foo@bar.com');
- console.log(res? chalk.green.inverse(res):chalk.red.inverse(res));
- **npm i nodemon -g.**
- **nodemon -verison.**
- **Module Wrapper Function in Node.JS.**
- js will wrap a module's code with a wrapper function before it is executed. The wrapper function looks like the following: Code: (function(exports, require, module, __filename, __dirname) { // Module code });
- (function(exports, require, module, __filename, __dirname){
-    const name = "Shanelhai";
-    console.log(name);
- });
- Node.js data is private because IIFE(immediately Invoked Function Expression)
- **Important don’t use this same code in node.js because node.js module wrapper function in add.**

- **How to create a web server using node.js.**
- **Node.js web server.**
- To access web pages of any web application, you need a web server. The web server will handle all the http requests for the web application.
- IIS is a web server for ASP.NET web application and Apache is a web server for PHP or JAVA web application.
- Node.js provide capabilities to create your own web server which will handle HTTP requests asynchronously. You can use IIS or Apache to run Node.js web application but it is recommended to use Node.js web server.
- The http.createServer() method includes request and response parameters which is supplied by node.js.
- The request object can be used to get information about the current HTTP request ex. url, request header, and data.
- The response object can be used to send a response for a current HTTP request.
- If the response form the HTTP server is supposed to be displayed as HTML, you should incude an HTTP header with the correct content types.
- **Code node.js server index.js.**
- const http = require("http");
- const server = http.createServer((req, res)=>{
-     res.end("Hello Shanelhai Saifi");
- });
- server.listen(8000, "127.0.0.1",()=>{
-     console.log("Listen to the port no 8000");
- });

- **How to handled routing and HTTP request.**
- **Index.js.**
- const http = require("http");
- const server = http.createServer((req, res) => {
-     if (req.url === "/") {
-         res.end("Hello Shanelhai Saifi home side");
-     } else if (req.url === "/about") {
-         res.end("Hello Shanelhai Saifi about side");
-     } else if (req.url === "/contactUS") {
-         res.end("Hello Shanelhai Saifi ContactUS side");
-     } else {
-         res.writeHead(404, { "Content-Type": "text/html" });
-         res.end("<h1>404 error page. Page does not exist</h1>");
-     }
- });
- server.listen(8000, "127.0.0.1", () => {
-     console.log("Listening on port 8000");
- });

- **JSON in Node.js.**
- JOSN stands for JavaScript object Notation JSON is a lightweight format for storing and transporting data. JSON is often used when data is sent from a server to a web page.
- **Code in index.json.**
- //object.
- const bioData = {
-     name: "Shaan",
-     age : 21,
-     fulln: "Shanelhai Saifi",
- };
- **convert object in json.**
- const jsonData = JSON.stringify(bioData);
- console.log(jsonData);
- **json convert in object.**
- const objData = JSON.parse(jsonData);
- console.log(objData.fulln);
- **File system.**
- const fs = require('fs');
- //object.
- const bioData = {
-     name: "Shaan",
-     age : 21,
-     fulln: "Shanelhai Saifi",
- };
- **Object Convert to JSON.**
- const jsonData = JSON.stringify(bioData);
- **Create a file add this data.**
- fs.writeFile("json1.json",jsonData,(err)=>{
-     console.log("done");
- });
- **Read this data and convert again object.**
- fs.readFile("json1.json", "utf-8",(err, data)=>{
-     const orgData = JSON.parse(data);
-     console.log(data);
-     console.log(orgData);
- });

- **How to create an API in Node.js.**
- API is the acronym for application programming interface which is a software intermediary that allow tow application to talk to each other. Each time you use an app like Facebook send an instant message or check the weather on your phone, you are using an API.
- It allows to software communicate with each other.
- API is a service which allows as to request data.
- **Userapi.json.**

```
• const http = require("http");
• const fs = require("fs");
• const server = http.createServer((req, res) => {
•   const data = fs.readFile(`${__dirname}/userapi/userapi.json`,
"utf-8");
•     const objData = JSON.parse(data);
• 
•   if (req.url === "/") {
•       res.end("Hello Shanelhai Saifi home side");
•                 } else if (req.url === "/about") {
•                   res.end("Hello Shanelhai Saifi about side");
•                 } else if (req.url === "/contactUS") {
•                   res.end("Hello Shanelhai Saifi ContactUS side");
•         }else if (req.url === "/userapi") {
•              //      fs.readFile(`${__dirname}/userapi/userapi.json`,
"utf-8",(err,data)=>{
•              //    console.log(data);
•                // //data selection one by one
•                // const objData = JSON.parse(data);
•                 res.end(objData[1].name);
•                //});
•                 } else {
•                   res.writeHead(404, { "Content-Type": "text/html"
});
•                   res.end("<h1>404 error page. Page does not
exist</h1>");
•                 }         });
•           server.listen(8000, "127.0.0.1", () => {
•                console.log("Listening on port 8000");     });
```

- **Event module in Node.js.**
- Node.js has a built-in module, called “Events” where you can create-, fire-, and listen for -your own events.
- Registering for the event to be fired only one time using once.
- Create an event emitter instance and register a couple of callbacks.
- **Index.js.**
- const EventEmitter = require("events");
- const event = new EventEmitter();
- event.on("SayMyName",()=>{
-     console.log("your name is Shanelhai");
- });
- event.on("SayMyName",()=>{
-     console.log("your name is Saifi");
- });
- event.on("SayMyName",()=>{
-     console.log("your name is Wolf");
- });
- event.emit("SayMyName");
- **Registering for the event with call back parameters.**
- const EventEmitter = require("events");
- const event = new EventEmitter();
-
- event.on("CheckPage",(sc, msg)=>{
-     console.log(`status code is ${sc} and the page is ${msg}`);
- });
- event.emit("CheckPage", 200, "ok");

- **Streams and Buffer Node.js.**
- Streams are objects that let you read data from a source or write data to a destination in continuous fashion. In Node.js there are four types of streams.
- Streaming means listening to music or watching video in ‘real time’, instead of downloading a file to your computer and watching it later.
- **Readable =** Stream which is used for read operation.
- **Writable =** Stream which is used for write operation.
- **Duplex** = Stream which can be used for both readd and write operation.
- **Transform =** A type of duplex stream where the output is computer based on input.
- Each type of stream is on Event Emitter instance and throws several events at different instance of times. For example, some the commonly used events are.
- **Data =** This event is fired when there is data is available to read.
- **End =** This event is fired when there is no more data to read.
- **Error =** This event is fired when there is any error receiving or writing data.
- **Finish =** This event is fired when all the data has been flushed to underlying system.
- **Index.js**

```
const fs = require('fs');
const http = require('http');
const server = http.createServer();
server.on('request', (req, res) => {
   const rstream = fs.createReadStream("read.txt");
   rstream.on('data', (chunkdata) => {
      res.write(chunkdata);                 });
   rstream.on('end', () => {
      res.end();                            });
   rstream.on('error', (err) => {
      console.log(err);
      res.writeHead(404, { 'Content-Type': 'text/plain' });
      res.end("file not found");            });                        });
server.listen(8000, "127.0.0.1", () => {
   console.log("Listening on port 8000");
});
```

- **Advance in one line code than 20 lines of stream.**
- **Method name pipe().**
- Stream.pipe(), the method used to take a readable stream and connect it to a writeable stream.
- **Index.js**

```
const fs = require('fs');
const http = require('http');
const server = http.createServer();
server.on('request', (req, res) => {
   const rstream = fs.createReadStream("input.txt");
   rstream.pipe(res);
});
server.listen(8000, "127.0.0.1", () => {
   console.log("Listening on port 8000");
});
```

- **Send email nodejs.**
- NodeMailer is a Node JS module that allows you to send emails from your server easily. It is a zero dependency module for all Node JS-compatible applications. The emails sent can be plain text, attachments, or HTML.
- **https://ethereal.email.**
- **https://nodemailer.com.**
- **Index.js**

```
const http = require("http");
const nodemailer = require("nodemailer");
const server = http.createServer((request, response) => {
   const transporter = nodemailer.createTransport({
      service: "gmail",
      secure: true,
      port: 465,
      auth: {
         user: "shanelhai9@gmail.com",
         pass: "Shaan@123"               }               });
   const mailOptions = {
      from: "shanelhai9@gmail.com",
      to: "shanelhai7@gmail.com",
      subject: "Node.js Mail Testing!",
      text: "Hello, this is a text mail!"          };
   transporter.sendMail(mailOptions, (error, info) => {
      if (error) {
         console.error("Error sending email:", error);
         response.writeHead(500, { 'Content-Type': 'text/plain' });
         response.end('Error sending email');
         return;               }
      console.log("Email sent successfully:", info.response);
      response.writeHead(200, { 'Content-Type': 'text/plain'
});
      response.end('Email sent successfully');
});
});
server.listen(8000, () => {
   console.log("Server is listening on port 8000");
});
```

Express JS

- **Express JS Basic.**
- Express JS is a Node JS web application server framework, designed for building single-page, multi-page, and hybrid web application.
- Several popular Node.js framework are built on Express, Ex. Nestjs, feathers etc.
- Try to write a small Rest API server in plain Node JS (that is, using only core modules) and then in Express JS. The latter will take you 5-10x less time and lines of code.
- **automaticall server run nodemon.**
- https://www.npmjs.com/package/nodemon.
- **Express web site.**
- https://expressjs.com/en/starter/installing.html.
- **Express.js**
- const express = require("express");
- const bodyParser = require("body-parser");
- const path = require("path");
- const app = express();
- app.use(bodyParser.urlencoded({ extended: true }));
- // Route handlers
- app.get("/", (req, res) => {
-     res.send("Welcome to Home page");
- });
- app.get("/about", (req, res) => {
-     res.send("Welcome to About page");
- });
- app.get("/services", (req, res) => {
-     res.send("Welcome to services page");
- });
- app.get("/contactus", (req, res) => {
-     res.send("Welcome to Contact page");
- });

- **Serve HTML file**
- app.get("/calculator", (req, res) => {
-     res.sendFile(path.join(__dirname, "index.html"));
- });
- **Post.**
- To handle http post request in Express.js version 4 and above, you need to install middleware module called body-parser.
- body-parser extract the entire body portion of an incoming request stream and exposes it on req.body
- The middleware was a part of Express.js earlier but now you have to install it separately.
- This body-parse module parses the JSON, buffer, string and URL encoded data submitted using HTTP POST request. npm install body-parser using npm as shown below.
- app.post("/calculator", (req, res) => {
-     console.log(req.body);
-     let n1 = parseFloat(req.body.v1); // Convert to number
-     let n2 = parseFloat(req.body.v2); // Convert to number
-     let sum = n1 + n2;
-     res.send("The sum of the two numbers is: " + sum);
- });
- **Start the server**
- app.listen(8000, () => {
-     console.log("Server is running on port 8000");
- });

Express.js

- **Express JS.**
- Express JS a Node JS framework. It’s the most popular framework as of now (the most starred on NPM).
- Express JS is a web application framework that provides you with a simple API to build websites, web apps and back ends.
- **Why do we actually need Express.js?**
- It is used to build a single page, multipage, and hybrid web application.
- **How it is useful for us to use with Node.js?**
- Try to write a small REST API server in plain Node.js (that is, using only core modules) and then in Express.js. The latter will take you 5-10x less time and lines of code.
- Creates an Express application. The express() function is a top-level function exported by the express module.
- **API CRUD.**
- GET = READ
- POST = CREATE
- PUT = UPDATE
- DELETE = DELET
- **Index.js**
- const express = require('express');
- const app = express()
- app.get("/", (req, res)=>{
-     res.send("Hello Shanelhai Saifi");
- });
- app.listen(8000,()=>{
-     console.log("listing the port at 8000");
- });

- **Website Routing.**
- Routing or router in web development is a mechanism where HTTP requests are routed to the code that handles them. To put simply, in the Router you determine what should happen when a user visits a certain page.
- **Index.js.**
- const express = require("express");
- const app = express();
-
- app.get("/", (req, res)=>{
-     res.send("Hello Welcome to the Home Page");
- });
- app.get("/about", (req, res)=>{
-     res.send("Hello Welcome to the About Page");
- });
- app.get("/services", (req, res)=>{
-     res.send("Hello Welcome to the Service Page");
- });
- app.get("/contact", (req, res)=>{
-     res.send("Hello Welcome to the Contact Page");
- });

- app.listen(8000,()=>{
-     console.log("listing the port at 8000");
- });

- **How to send HTML and JSON data as a response using express.**
- The method are identical when an object or array is passed,but res.json() will also convert non-objects, such as null and undefined, which are not valid JSON.
- **Index.js.**
- const express = require("express");
- const app = express();
- //HTML code.
- app.get("/", (req, res)=>{
-     res.write("<h1>Hello Welcome to the Home Page</h1>");
-     res.write("<h1>Hello Welcome to the agan Home Page</h1>");
-     res.send();
- });
- app.get("/about", (req, res)=>{
-     res.send("Hello Welcome to the About Page");
- });
- **Array object code.**
- app.get("/services", (req, res)=>{
-     res.send([{
-         id:1,
-         name: "Shanelhai",
-     },
-     {
-         id:1,
-         name: "Shanelhai",
-     },
-]);   });
- **JSON Data code.**
- app.get("/contact", (req, res)=>{
-     res.send({
-         id:1,
-         name: "Shanelhai",
-     });   });
- app.listen(8000,()=>{
-     console.log("listing the port at 8000");
- });

- **Serve HTML CSS & JS Files in Express JS.**
- **Serving Static files in Express.**
- To serve static files such as images, CSS files, and JavaScript files, use the express.static built-in middleware function in Express.
- **Index.js.**
- const path = require("path");
- const express = require("express");
- const app = express();
- **HTML and CSS and Files server code.**
- const staticpath = path.join(__dirname, "../public");
- app.use(express.static(staticpath));
-     app.get("/", (req, res)=>{
-             res.send("Hello Welcome to the Home Page");
-         });
-         app.get("/about", (req, res)=>{
-       res.send("Hello Welcome to the About Page");
-         });
-         app.get("/services", (req, res)=>{
-             res.send("Hello Welcome to the Service Page");
-         });
-         app.get("/contact", (req, res)=>{
-             res.send("Hello Welcome to the Contact Page");
-         });
-         app.listen(8000,()=>{
-             console.log("listing the port at 8000");
-         });
- **How to create static web site and serve the web site express js.**
- const path = require("path");
- const express = require("express");
- const app = express();
- //HTML and CSS and Files server code.
- const staticpath = path.join(__dirname, "../public");
- app.use(express.static(staticpath));
- app.get("/", (req, res)=>{
- res.send("Hello Welcome to the server");   });
- app.listen(8000,()=>{
- console.log("listing the port at 8000");});

- **Template Engines (Pug, hbs, EJS) in Node JS Add Dynamic Content.**
- hbs = npm install hbs
- Express.js view engine for handlebars.js
- Make folder name views mkdir
- **Index.js**
- const path = require("path");
- const express = require("express");
- const app = express();
- **Define the path to your static files**
- const staticPath = path.join(__dirname, "../public");
- **Set the view engine (assuming you're using Handlebars here; change if needed)**
- app.set("view engine", "hbs");
- **Define the path for static files**
- app.use(express.static(staticPath));
- **Route to render the view**
- app.get("/", (req, res) => {
-   res.render("index", {
-     channelName: "Shanelhai",
-   });
- });
- app.get("/",(req, res)=>{
-         res.send("Hello The server is start");
- });
- **Start the server**
- app.listen(8000, () => {
-   console.log("Listening on port 8000");
- });
- **Index.hbs**
- <li><a href="#">{{channelName}}</a></li>

- **Customizing Views Directory.**
- To change the views directory, use the app.set() method.
- **Index.js**
- const path = require("path");
- const express = require("express");
- const app = express();
- Define the path to your static files
- const staticPath = path.join(__dirname, "../public");
- Views change.
- const templatePath = path.join(__dirname, "../templates");
- Set the view engine (assuming you're using Handlebars here; change if needed)
- app.set("view engine", "hbs");
- app.set("views",templatePath);
- Define the path for static files
- app.use(express.static(staticPath));
- Route to render the view
- app.get("/", (req, res) => {
-   res.render("index", {
-     channelName: "Shanelhai",
-   });
- });
- app.get("/",(req, res)=>{
-         res.send("Hello The server is start");
- });
- Start the server
- app.listen(8000, () => {
-   console.log("Listening on port 8000");
- });

- **Partials in Express.js**
- Express. js allows the creation of partial views, which are essentially smaller, reusable components of a larger view. These partials can represent specific sections of a webpage, such as headers, footers, or sidebars. Including a partial in a view is achieved through a simple syntax, enhancing code organization.
- **Index.js**
- const path = require("path");
- const express = require("express");
- const app = express();
- const hbs = require("hbs");
- Define the path to your static files
- const staticPath = path.join(__dirname, "../public");
- Define the path to your templates
- const templatePath = path.join(__dirname, "../templates/view");
- Define the path to your partials
- const partialsPath = path.join(__dirname, "../templates/partials");
- Set the view engine to Handlebars
- app.set("view engine", "hbs");
- app.set("views", templatePath);
- Register the path to your partials
- hbs.registerPartials(partialsPath);
- Serve static files from the "public" directory
- app.use(express.static(staticPath));
- Route to render the index view
- app.get("/", (req, res) => {
-   res.render("index", {
-     channelName: "Shanelhai",
-   });   });
- Start the server
- app.listen(8000, () => {
-   console.log("Server is listening on port 8000");
- });
- **Header.hbs**
- {{>header}}

- **Adding 404 error page in express.**
- **Index.js**
- const path = require("path");
- const express = require("express");
- const app = express();
- const hbs = require("hbs");
- **// Define the path to your static files.**
- const staticPath = path.join(__dirname, "../public");
- **// Define the path to your templates.**
- const templatePath = path.join(__dirname, "../templates/view");
- **// Define the path to your partials.**
- const partialsPath = path.join(__dirname, "../templates/parsial");
- **Set the view engine to Handlebars.**
- app.set("view engine", "hbs");
- app.set("views", templatePath);
- **Register the path to your partials.**
- hbs.registerPartials(partialsPath);
- **Serve static files from the "public" directory.**
- app.use(express.static(staticPath));
- **Route to render the index view.**
- app.get("/", (req, res) => {
-   res.render("index", {
-     channelName: "Shanelhai",
-   });   });
- app.get("/about",(req, res)=>{
-   res.render("about");
- });
- app.get("*", (req, res)=>{
-   res.render("Hello for the 404 error page",{
-     errorcomement:"Opps page could't be found",
-   });   });
- **Start the server.**
- app.listen(8000, () => {
-   console.log("Server is listening on port 8000");
- });

- **Folder Structure express JS.**
- my-express-app/
- │
- ├── public/
- │ ├── css/
- │ ├── js/
- │ ├── images/
- │ └── uploads/
- │
- ├── src/
- │ ├── controllers/
- │ ├── middleware/
- │ ├── models/
- │ ├── routes/
- │ └── services/
- │
- ├── views/
- │ ├── partials/
- │ ├── layouts/
- │ └── index.ejs
- │
- ├── .env
- ├── app.js
- ├── package.json
- └── README.md
- **API.**
- **Npm I requests.**
- **Same as Node JS work API.**

MongoDB

- **MongoDB.**
- MongoDB is a non-relational, open-source database management system (DBMS) that stores data in documents instead of tables and rows. It's designed for modern application development and the cloud.
- **Here are some of MongoDB's features:**
- Document-oriented: MongoDB stores data in flexible, JSON-like documents that can vary from document to document.
- Scalable: MongoDB's scale-out architecture allows you to add more nodes to share the load as demand increases.
- Flexible: MongoDB is built for high availability, horizontal scaling, and geographic distribution.
- Query API: MongoDB offers an expressive query API for working with data.
- Semantic search: MongoDB allows you to prompt LLMs with semantic search across vectors.
- Aggregation: MongoDB allows you to aggregate, transform, and analyze data in place.
- **SQL VS MongoDB.**
- **SQL.**
- RDBMS is a relational database management system and works on relational database.
- It stores data in form of entity as tables.
- It uses SQL to query database.
- **MongoDB.**
- It is a non-relational document-oriented database management system and works on document-based database.
- MongoDB stores data in form of document .
- MongoDB uses BSON to query database.

- **MongoDB CRUD Operations.**
- CRUD operations.
- C = create.
- R = read.
- U = update.
- D = delete.
- **Create Operation.**
- Create or insert operations add new documents to a collection.
- **Inser one data in collection.**
- db.inventory.insertOne({
-     item: 'canvas',
-     qty: 100,
-     tags: ['cotton'],
-     size: { h: 28, w: 35.5, uom: 'cm' }
- });
- **Inser many data in collection.**
- db.inventory.insertMany([
-     {
-       item: 'journal', qty: 25, size: { h: 14, w: 21, uom: 'cm' }, status: 'A' },
-     {
-       item: 'notebook', qty: 50, size: { h: 8.5, w: 11, uom: 'in' }, status: 'A' },
-     {
-       item: 'paper', qty: 100, size: { h: 8.5, w: 11, uom: 'in' }, status: 'D' },
-     {
-       item: 'planner', qty: 75,
-       size: { h: 22.85, w: 30, uom: 'cm' }, status: 'D' },
-     {
-       item: 'postcard', qty: 45, size: { h: 10, w: 15.25, uom: 'cm' }, status: 'A' }]);

- **Read Operation.**
- **Use of find method Select all documents.**
- db.inventory.find()
- **Select one document.**
- db.inventory.find({qty:80})
- **Specify Conditions Using Query Operators.**
- db.inventory.find({status: { $in: ['A', 'D'] }})
- **Specify AND Conditions**
- db.inventory.find({status: 'A' , qty: { $lt: 30 }});
- **Specify OR Conditions.**
- db.inventory.find({ $or: [{ status: 'A' }, { qty: { $lt: 30 } }]});
- **Update Documents Operation.**
- **Update a Single Document**
- db.inventory.updateOne(
-   { item: 'paper' },
-   { $set: { 'size.uom': 'cm', status: 'P' },
-     $currentDate: { lastModified: true }
-   });
- **Update Multiple Documents.**
- db.inventory.updateMany(
-   { qty: { $lt: 50 } },
-   {
-     $set: { 'size.uom': 'in', status: 'P' },
-     $currentDate: { lastModified: true }
-   }
-);
- **Replace a Document**
- db.inventory.replaceOne(
-   { item: 'paper' },
-   {
-     item: 'paper',
-     instock: [
-       { warehouse: 'A', qty: 60 },
-       { warehouse: 'B', qty: 40 }
-     ]
-   }
-);

- **Delete Documents Operation.**
- **Delete All Documents.**
- db.inventory.deleteMany({});
- **Delete All Documents that Match a Condition.**
- db.inventory.deleteMany({ status: 'A' });
- **Delete Only One Document that Matches a Condition.**
- db.inventory.deleteOne({ status: 'D' });

- **Sort data.**
- Find a data one.
- db.inventory.find().limit(1);
- Skip the first document a data.
- db.inventory.find().skip(1);
- Sort the data ascending order.
- db.inventory.find().sort({qty:1});
- Sort the data descending order.
- db.inventory.find().sort({qty:-1});

- **Achieving Pagination using MondoDB find and limit.**
- no = 8
- db.blogs.find().skip((pageNo-1)*no).limit(no)
- pageno = 1 db.blogs.find().skip(0).limit(8)
- pageno = 2 db.blogs.find().skip(8).limit(8)

- **Mongodb Operators.**
- **Mongodb Query and Projection Operators.**
- db.inventory.find({ qty: { $lt: 30 } })
- **$in operator.**
- db.inventory.find({ qty: { $in: [25, 30] } })

- **Aggregation Pipeline.**
- db.orders.aggregate([

- Stage 1: Filter pizza order documents by pizza size
- {
- $match: { size: "medium" }
- },

- **Stage 2: Group remaining documents by pizza name and calculate total quantity**
- {
- $group: { _id: "$name", totalQuantity: { $sum: "$quantity" } }
- }

-])

- **Mongodb atlas**
- **Connecting with MongoDB Compass.**
- mongodb+srv://shanelhai7:<db_password>@cluster0.pipvr.mongodb.net/
- **password**
- MlM8x8aBXzqGfMDZ.