***- Backend is all about serving files to the clients from server***

**Node.js**

**Introduction**

- Until Node.js introduced , javascript was only a client-side language and it was executed only on Browsers

- Node.js is a javascript Runtime Environment built on **Chrome’s V8 Javascript engine** , which lets Javascript to run on server-side as well .

- V8 is Google’s open source high-performance JS engine written in C++ , used in chrome and node.js

- V8 is the thing that takes out JS and executes on Chrome or It provides the run-time environment for JS

- Runtime Environment means , the environment in which a program or application is executed . Ex:- For JS back then browsers were only the runtime , but now local system too has become runtime for JS .

- Node.js is designed to built scalable network applications

- It is free and open source server environment

- It can run on multiple OS

- It allows us to run JS on the server or outside of the browser .

**Why use Node.js**

- By Node.js , we can use JS in the entire stack

- Many famous companies use it as their backend

- It comes with lots of useful built-in modules

- It is very fast, lightweight and efficient

**REPL(Read-Evaluate-Print-Loop)**

- REPL is an interactive shell that processes Node.js expressions and it is activated in terminal on writing ‘node’ command

- In REPL underscore or ‘**\_**’ holds the previous variable

- On hitting tab bar twice in REPL all the global variables/modules will be printed and these variables are always available to use and need not to be imported in a node.js file

**Modules/Packages**

- Basically , module is blocks of encapsulated codes written by other coders and can be used by any user simply by Importing them in our file , in order to achieve any functionality .

- In node.js , Modules are the blocks of encapsulated code that communicates with an external application on the basis of their functionality .

- Modules can be a single file or can be a collection of multiple files , these files consist methods , classes and other relevant stuffs .

**Types of modules in Node.js**

1 . Core modules / built-in modules

- These are the built-in modules that are the part of the platform and comes with Node.js installation .

- These modules can be loaded/imported into the program by using the **require ()** method .

ex: - http - creates an HTTP server in Node.js

fs - used to handle file-system

process - provides information and control about the current Node.js process .

os - provides information about the operating system. etc

2 . Local modules/User-defined modules

- Unlike built-in and external modules , local modules are creates locally by the user in the Node.js application . One can create his own local module . Ex:- Let’s create our own calc.js module

Filename: calc.js

exports.add = function (x, y) {

return x + y; };

exports.sub = function (x, y) {

return x - y; };

exports.mult = function (x, y) {

return x \* y; };

exports.div = function (x, y) {

return x / y; };

3 . Third-party Modules

- Third-party modules are modules that are available online and one can access it using the Node Package Manager(NPM)

- These modules can be installed in the project folder of globally

ex:- mongoose(npm install mongoose) , express(npm install express) , angular(npm install -g@angular/cli) and react

**const fs = require("fs");**

- It returns an object

- require(“module\_name”) is an **include-module** used to include or load JS-modules .

- require() module reads a JS-file , executes the file , and then proceeds to return the **exports object**.

- **fs** module stands for **file-system** module which allows you to work with the file system on our computer .

**const text = fs.readFileSync(“text\_file\_name” , “encoding”) -** This method is an inbuilt application programming interface of **fs** module which is used to read the file and return its content .

- utf-8(Unicode Transformation Format) is preferred encoding for e-mail and web-pages

**text = text.replace("fun","run");** -This method replace run with fun , the text file

**fs.writeFileSync("text\_new.txt",text);** - This will write the content of text into text\_new.text file , and here we don’t need to store the statement in a variable

**Synchronous(Blocking) and Asynchronou(non-blocking) code/calls**

|  |  |
| --- | --- |
| **Blocking code(Synchronous)** | **Non-Blocking code(Asynchronous)** |
| As name suggests , it means to be in a sequence ,  Every statement of code gets executed line by line | These methods execute asynchronously i.e It gets  executed line by line but not in every case . |
| Every line of code waits for its previous one to get executed first and then it gets executed | If any error occurs , these codes allow the program  to be executed further immediately without any  blocking . |
| Hence , If an error occurs in a line , the whole code  will be blocked until that error is fixed | The asynchronous functions requires a callback  method inside it as an argument to deal with error  in background letting rest of the code to get  executed. The callback will fire only if the job of the  non-blocking function is done . |
| Ex:- readFileSync(“file\_name” ,”encoding”) | Ex:- readFile(“file\_name” , “encoding” ,  callback-function) |
| The blocking counterparts of non-blocking  methods have names ending with **Sync** | All of the I/O methods in Node.js standard library  provide asynchronous versions , which are  non-blocking and accept callback functions |

Blocking code:-

const fs = require('fs');

const data = fs.readFileSync('/file.md'); **// blocks here until file is read**

console.log(data);

moreWork(); **// will run after console.log**

Non-Blocking code:-

const fs = require('fs');

fs.readFile('/file.md', **(err, data) => { //callback method**

**if (err) throw err;**

**console.log(data);**

}); **// this won’t block the program and will be running in the background until the file is read or until the error is detected**

moreWork(); **// will run before console.log**

**- Node.js works on Non-Blocking I/O model**

**Creation of simple sever which shows nav.html on server**

**const http = require('http');**

**const fs = require('fs');**

**const file = fs.readFileSync('nav.html');**

**const server = http.createServer((req , res)=>{**

**res.writeHead(200 , {'Content-type':'text/html'});**

**res.end(file)**

**})**

**server.listen(80 , '127.0.0.1',()=>{**

**console.log("Listen on port 80")**

**})**

- **‘http’** is a buit-in module which allows Node.js to transfer data over the **Hyper Text Transfer Protocol(HTTP)**

- HTTP is a communications protocol which is used to send and receive webpages and files on the internet

- It is a standatrd application-level protocol used for exchanging files on the World Wide Web

- It is a client-server protocol , which means requests are initiated by the recipient , which is usually aWEB- Browsers

- Clients and Servers communicate by exchanging individual messages .

- The messages sent by the client (WEB-BROWSER) are called **HTTP-requests** and the messages sent by the server as an answer are called **HTTP-responses** .

-It is the protocol designed to enable communication between client and servers

http.**createServer(requestListener)**

- createServer method used to create a server on your computer

- It turns your computer into a HTTP server

- **http.createServer()** creates an **HTTP Server object**

- the HTTP server object can listen to the ports on your computer and execute a function **, requestlistener** , each time a request is made .

- **requestListener** : It specifies a function to be executed every time the server gets request . This function is called a requestListener , and handles request from the user , as well as reponse back to the user

- **req.url** gives the requested url from client side.

- **response.writeHead(status code , {‘Content-Type’:’text/plain’})** is an inbuilt property of the ‘http’ module which sends a response header to the request

- **response.setHeader()** allows you to set only a singular header and won’t take status code as its argument unlike writeHead() .

- **status code 200** means that all is OK , 404 is page not found

- **port** is a number used to uniquely identify a transaction over a network by specifying both the host and the services . Port is the address of the service/transaction between the client and server . POrt 80 and 443 are the two ports used by web-servers to communicate with the web-clients

- **res.end(data)** : It is used to end the response process .

- **server.listen(port , hostname , backlog, callback)** : It creates a listener(which listens the client request) on the specified port or path

- **hostname** specifies the IP address we want to listen to

- Specifies a function to be executed when the listener has been added

**Creating our own module**

index.js

const mod = require('./mod');

console.log**(mod([4,5])**); **// mod.average([4,5]) won’t work here**

mod.js (module-file)

console.log("This is my custom module");

function average(arr){

sum=0;

arr.forEach(element => {

sum += element;

});

return sum/arr.length;

}

**//** **In node.js we have to explicitly mention the methods or classes that we want to export to other files**

module.exports = average**; // we must mention this line to every method to be export**

**NPM (Node Package Manager)**

- It is the default package manager for the Node.js

- It is used to install modules , packages/modules in our project

- **npm-init** : It initializes your project-folder as a node-package and it installs package.json . By this whenever we install any module like express , mongoose , slugify etc to the project , node will include them in dependencies section of package.json.

- **package-json** : This file holds metadata relevant to the project . This file is used to give information to npm that allows it to identify the project as well as handle the project's dependencies.

- **dependencies** : are the packages/modules on which packages.json/our-project is depenedent.

- One package is dependent on its dependencies and these dependencies can be dependent further on other multiple packages/dependencies and so on . ex: express , mongoose etc

- **devDependencies** : These are the modules which are only required during the development and not on runtime or production , whereas dependencies are required at runtime as well. Ex:- Nodemon , Babel , Chai etc

- Hence , the dependencies needs to be installed in the project while deploying it online without any devdependencies.

- **node-modules** folder is the folder where all the projects related modules/packages are kept

- we can install any specific version of a module we use **npm install** [**slugify@1.3.5**](mailto:slugify@1.3.5)

here in **1.3.5** , 1 - major version(major-changes) , 3 - minor version(small-changes) , 5-patch (bug-fixes)

“dependencies”:{ “package1”: “**^**16.5.0 “, “package2”:”**~**1.3.3” , “package”: ”**>**1.0.1”}

if any package is deleted from node-modules folder , then they can be retrieved by npm i and ,

**^** : this means exactly the given version will be installed by npm

**~** : This means npm will install the version with **newer-patch** if any available

**>** : this means if any new **major-version** is available then that will be installed / or simply the latest version will be installed

**Nodemon**

- It is a tool and a command-line interface(CLI) utility that helps to develop node.js based applications by automatically restarting the node application when the file changes in the dirtectory are detected

**npm i/install nodmon** : To install nodemon

**npm i/install -g nodemon** : To install nodemon globally on our machine(we can use nodemon in any project on our system)

**npm i/install nodemon** --save-dev : Install nodemon on your project as dev-dependency

**Postman**

- It is software testing API platform to build , test , design , modify , and document APIs .

- This tool has ability to make various types of HTTP requests like GET , POST , PUT , PATCH .

- And it also converts the API to code for languages like Jasvascript and Python

**Express.js**

- It is a minimal and flexible Node.js web application framework/package/module/library that provides s robust set of features for web and mobile applications .

- It is a free and open-source web application framework for Node.js.

- It used for designing and building web-applications and APIs quickly and easily

- It is a faster server-side framework

- Express.js reduces the coding-time and effort to build and organize any web-app

***Features , advantages and differences regarding express , node and anguar:*** [***https://www.besanttechnologies.com/what-is-expressjs***](https://www.besanttechnologies.com/what-is-expressjs)

**HTTP types of Messages and request-Methods and status-codes**

Types of messages :

request - send by the client to trigger an action on the server

response - send by the server to to client as per the request

Types of request-Methods:

GET - This method is used to request data from specified resource on server .

POST - This method is used to send data to a server and request to create/update a resource

Status-Codes

200 : Ok

404 : Page not found

**Routing**

**-** It refers to determining how an application responds to a client request to a particular end-point , which is URL(or path) and a specific HTTP request method

ex : app.METHOD(PATH , HANDLER(callback fun)) ,where

- app is an instance of express .

- METHOD is an HTTP request-method , in lower case i.e get() , post()

- PATH is path on the server i.e ‘/’ , ‘/about’ etc

- HANDLER is the function executed when the route is matched

***more on routing and route-paths : https://expressjs.com/en/guide/routing.html***

**First Express-app**

const express = require('express');

const app = express();

const port = 80;

**// following will respond with "this is my first-express app" when a GET request is made to the homepage or ‘/’ end-point**

app.get("**/**" , (req,res)=>{ **//it will handle GET request on ‘/’ end-point**

res.send("this is my first-express app")

});

**// following will respond with " this is About section of my first-express app " when a GET request is made to the ‘/about’ end-point**

app.get("**/about**" , (req,res)=>{

res.send("this is About section of my first-express app")

});

**// following will respond with " this is post request of about section of my first-express app " when a POST request is made to the ‘/about’ end-point**

post.get("**/about**" , (req,res)=>{ **//it will handle POST request on ‘/about’ end-point**

res.send("this is post request About section of my first-express app")

});

app.listen(port , ()=>{

console.log(`This app is running on port ${port}`);

});

where ,

‘**/**’ - root path/route or the app’s homepage / local host

- port 80 works without even mentioning it , that’s we used it .

- **app.listen()** function is used to bind and listen the connections on the specified host and port .

- ‘**/**’ and ‘**/about**’ both are end-points or paths

- app.get() is the express.js specific feature and it lets us define a route-handler(callback fun) for GET requests to a given URL/path . And this method also prevents us to write numbers of code for the same usage

**Endpoint** : When an API interacts with another system , the touchpoint of this communication are termed as endpoints . Like , URL of a server , localhost

- app.get() hadles GET-request and app.post() handles POST-request at any end-point like ‘/’ , ‘/about’

- res.**status(200)**.send("this is my first-express app") , this will throw status code too , along with the reponse

**Static files**

-These files are any content that can be delivered to the user without having to be generated , modified or processed .

- It will be available to the user as it is without any dynamic-action applied to it

- These files don’t change when your application is running

- **Express is a routing and middleware web framework that has minimal functionality of its own**

- An express application is essentially a series of middleware function calls

**Middleware**

- It is anything you put in the middle of one layer of the software and another Or , It is those methods/functions/operations that are called between processing the Request and sending the Response in our application method .

- Middleware functions are functions that have access to the request object(req) , the response object(res),

and the next function in the application’s request-response cycle

- The **next** function is a function in the EXpress router which , when invoked , executes the middleware which is succeeding(upcoming) the current middleware .

ex:- app.get("**/about**" , (req,res)=>{

res.send("this is About section of my first-express app")

}); //**In this router the second argument is middleware function**

- To serve a static files like , CSS files , JS files , images we use code : **app.use('/static', express.static('public'))**

- **use()** function is used to mount/link the specified middleware functions at the path which is being specified

syntax : app.use(path , callback)

- **app.use(path , middleware)** is basically used to mount/bind a middleware function on a path and to load it .

-**‘/static’** is path/endpoint

- **express.static()** is built-in middleware function in Express used serve static file . **‘public’** is directory to serve

**Template Engines**

- Template engines help us to create an HTML template with minimal code .

- At runtime it injects data into the HTML template or it replaces variablels in template file with actual values and finally it transforms the template into an HTML file which is further send to the client-side .

Ex:- Jade(Pug) , EJS , Vash , Mustache etc

**PUG.js**

- It is a template/view engine for Node and for the browser .

- A template engine compiles a template into HTML code .

- Pug has simpler syntax than HTML

syntax:- doctype html

html

head

style

include ../static/style.css

body

h1 My Site

p Welcome to my super lame site

script

include ../static/index.js

More Pug Templates:-

**doctype html**

**html**

**head**

**style** 🡨 including CSS file to PUG

**include ../static/style.css**

**body**

**nav#navbar** 🡨 setting id

**ul**

**div#logo**

**img(src="/static/logo4.png")**

**|HDA**

**li #[a(href="/") Home]** 🡨 using anchor tag

**li #[a(href="/") About]**

**li #[a(href="/") Contact us]**

**li #[a(href="/") Services]**

**li #[a(href="/") Class info]**

**li #[a(href="/") Missions]**

**section#introSection.intro**

**div#main Let us have a chance to grab out the best of yours**

**div#sub focus learn dance and bang the floor**

**section#missionSection**

**h4 Our Missions**

**div.cards**

**div.cardOne**

**h6 Dark perfection**

**div.card-box** 🡨 setting class

**div.card-image**

**img(src="/static/img1.png")**

**div.card-content**

**p This is the best card academy**

**div.cardOne**

**h6 Dance the way you like**

**div.card-box**

**div.card-image**

**img(src="/static/img2.png")** 🡨 Image insertion

**div.card-content**

**p This is the best card academy**

**div.cardOne**

**h6 Expert teaching**

**div.card-box**

**div.card-image**

**img(src="/static/img4.png")**

**div.card-content**

**p This is the best card academy**

**section#sponsorsSection.sponsors**

**| Our Sponsors 🡨 Inserting text using ‘|’ character without any tag**

**footer#footer**

**| Footer**

**script** 🡨 including JS file to PUG

**include ../static/index.js**

**\* PUG supports Template Inheritance**

- Template Inheritance allows you to build a base “skeleton” template that contains all the common elements of your site and defines blocks that child templates can override

- In PUG it works via **block** and **extends** keywords

- Template Inheritance prevents to write same piece of code in different files again and again .

**To render PUG template files in our express project we will do some settings as follows** ,

- Set the template-engine as Pug : **app.set(‘view engine’ , ‘pug’)** , it will load the template engine module

- Set the directory where the template files are located : **app.set(‘dir\_name’ , ‘dir\_address’)**

ex: app.set(‘views’ , path.join(\_\_dirname , ‘views’))

- create a Pug template file named index.pug in the views directory , with following code

html

head

title= title

body

h1= message

- Create a route to render the index.pug file at a particular end-point i.e /demo

app.get('/demo', function (req, res) {

res.render('index', { title: 'Hey', message: 'Hello there!' })

})

- **res.render()** function is used to render a view/template and sends the rendered item to the client . It is similar to res.send() but specificly used to send templates to client side

- **path-module** : This module provides utitlities for working with file and directory paths .

- **path.join(paths)** : It joins the specified path segments into one path . The specified path segments must be strings

ex: var x = path.join('Users', 'Refsnes', 'demo\_path.js');

console.log(x) ;

Output : Users\Refsnes\demo\_path.js

ex: console.log(path.join(\_\_dirname,'views')) , \_\_dirname holds current directory address

Output : e:\CareerAndWork\ComputerScienceAndcollege\codes\vscode\webD\new\Express\views

- We can create our own pug template with HTML

- To insert and use variables in our self-made template we can do like ,

**in pug file**

<title**>#{title}</**title>

<p>

**|#{content} 🡨 pipe character is important here , as it is not in between <p> horizontally**

</p>

**passing the variables values in node.js file**

res.status(200).render('index.pug' **, {'title':'pug is awesome','content':'hello boyis'});**

- To insert css file in our pug file , we can do like ,

<title>#{title}</title>

**style**

**include ../static/style.css** 🡨 it goes out of the current dir by ‘**..**’ and get into static dir to get style.css file

**Getting and saving/parsing client-side form data to our backend , if a form is filled up and makes a post request to the server**

we use following code ,

app.use(**express.urlencoded())**

app.post('/',(req,res)=>{

let name = **req.body ;**

**console.log**(name) ;

const params = {'message':'Your form has been submitted successfully'} ;

res.status(200).render('index.pug',params) ;

})

**express.urlencoded()**

- This function is a built-in middleware function in Express .

- It parses/recogniozes the incoming Request Object as strings or arrays

- It is based on body-parser

- To load this method , we go like , **app.use(express.urlencoded())**

- It returns an object

**req.body**

- It contains an object or key-value pair of data submitted in the request body / forms

- If we submit a form in a request body , then the object returned by req.body will contain the the **name attribute** of the input element **as keys** and the submitted **values** as the **values** , like ,

**{name\_one : submitted value , name\_two : submitted value }**

i.e , **body-form** ,

<form action="/" id="contact" method="post">

<input type="text" **name="name"** id="name" placeholder="Enter your name ">

<input type="text" **name="age"** id="age" placeholder="Enter your age ">

**returned object by req.body** ,

{

**name: 'Rahul Kumar',**

**age: '22',**

gender: 'Male',

address: 'Budhha Nagar - 1/A, Postal Park , Chirayiyatad , Patna - 01',

more: 'as'

}

**Mongodb**

- No SQL Database

- Document oriented

- Open-source , cross-platform , written in C++

- Salient features:-

- Develop faster

- Deploy Easier

- Scale Bigger

|  |  |
| --- | --- |
| **Relational DBMS** | **Object Oriented DBMS** |
| Stores data in separate tables and that are defined  by the programmer | These are NoSQL DBMS AND Stores data in the form of documents |
| It consists Database , Tables and Rows | It consists Database , Collections and Documents |
| Ex :- MySQL , Oracle etc | Data is stored as BSON(Binary Javascript object notation) |
|  | ex: MongoDb , DynamoDB etc |

|  |  |
| --- | --- |
| Mongo | Mongod(Mongo Daemon) |
| Mongo is a command -line shell that connects to  specific instance of mongod | Its basically the host process for the database ,  which starts the DBMS |
| With the help of Mongo we can connect the  mongod process hosted in another system | This means you’r basically saying “start the  MongoDB database process and run it in the  background” |
| mongo can be considered as client which connects  with mongod main db-process | mongod is the main process of mongodb database |

- When you first start mongoDB , it starts at 27017 port by-default

- **mongod** process takes the command from mongo-shell and stores them in data directory which we have created.

- mongo gives a uinque ID to every items

**BSON**

**-** In BSON data-types of variables are declared like , int , strings

- BSON is the binary encoding of JSON-like documents that MongoDB uses when storing documents in collections

|  |  |
| --- | --- |
| **JSON** | **BSON** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**\*\*head over to MongoDB documentation for more info about BSON and difference b/w BSON and JSON**

- By default mongodb uses **test** db

**Database schema**

- A database schema is a blueprint/format or architecture of how our data will look.

- It is the skeleton structure that represents the logical view of the entire database

- Schema is a logical reperesentation of data in database

ex: Schema of a student :

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Roll no** | **Class** | **Address** |

Schema of a Course :

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **price** | **duration** | **class** |

- In mongo a schema is a JSON object that defines the structure and content of your data .

- Data in MongoDB has flexible schema or you don’t need to keep same schema in all the docs.

**Mongo shell commands**

**CRUD :** It is an acronym that stands for **C**reate **R**ead **U**pdate **D**elete . These are the four more basic operations that can be performed with most traditional database systems and these are used to interact with any database

**C**reate

**use db\_name** : get you select db\_name database and if such db not present there then it will be auto-created

**db.text** : Creates a collection/table in the current/selected db , with name ‘text’

**db.text.insert({“name”:”Mongo”})** : Inserts the given document/row({“name”:”Mongo”}) to the collection named ‘text’ situated in the current/selected database

**show dbs** : shows all the Databases available with atleast 1 document

**show collections** : shows the collections of current/selected db

**db.items.insertOne(object)** : Used to insert single document/row in the current database **, here ‘items’ referes to a collection/table of the current db**

**db. items.insertMany([multiple objects]) :** Used to insert multiple documents/rows in the items collection/table of current db

**R**ead

**db.items.find()** : this command will show all the documents/rows of items collection/table of current db.

**db.items.find({rating:4.5})** : it will return only those filtered documents/rows that have rating as 4.5 . Here , **{rating:4.5}** is a **filter object**.

db.items.find({rating:**{$gte:3.5}**}) : It will return all the documents/rows having rating greater than or equal to 3.5 in items collection of the current database

We can implement **AND-operator** in filter object and for that we use **comma’,’** like below,

**db.items.find({rating: {$gt:3.5} , price:{$gt:4000}}) :** It will return the documents according to the filter objects

We can also implement **OR-operator** ,for that we use **$or** in the filter object. like below ,

db.items.find( { **$or**: **[**{rating:{$lt:3.5}}**,**{price:{$gt:114000}}**]** } ) , here we have used array of filter-objects

- We can use another argument apart from filter-objects that is called **projection** , like below ,

db.items.find( {rating:{$gte:3.4}} , **{rating:1}** ) : It will only show the ratings according to the filter-object

**D**elete

**db.items.deleteOne({price:22000}) :** deletes first document/row entry in case of multi-document match

**db.items.deleteMany(Multiple docs/{specific-doc}) :** Either deletes same-matching multiple docs or multiple different docs in one go

**U**pdate

db.collection.**updateOne**({selector-object},{**$set**:keys-to-update})

- It will update the key-values only of the first matched-document

- Two objects are passed , first one is **selector-object** , which selects the required document and second one is the key that we want to update .

**ex:- db.colOne.updateOne({name:"RBK"},{$set:{price:2,rank:1}})**

db.collection.**updateMany**({selector-object},{**$set**:keys-to-update})

- will update the key-values of all the matched docs

ex:- **db.colOne.updateMany({rank:22},{$set:{price:20202,rank:1}})**

**Mongoose and Mongodb-compass**

- Mongoose is a Node.js based Object Data Modeling (ODM) library for MongoDB , OR in simple words mongoose is the medium for Node.js to connect with the MongoDB

- Mongoose is a middle-layer thing between Node.js and MongoDB

- Mongoose is a cross-platform embedded web-server and networking library.

- MongoDB-compass is a GUI which replaces the monog-shell usage

- As we use mongo-shell to connect with mongoD main process of database , likewise we can connect to mongoD process using compass , python(using pymongo) , nodeJS(using mongoose) etc too.

First code using **mongoose**

const mongoose = require('mongoose'); **// Including mongoose module**

main().catch(err => console.log(err));

async function main() {

await mongoose.connect('mongodb://localhost:27017/test'**);//connecting to mongodb ‘test’ database at localhost**

}

const kittySchema = new mongoose.Schema({

name: String

**});// creating Schema in which we have set the name property as String**

const Kitten = mongoose.model(**'Kitten'**, kittySchema);**// getting the schema compiled and getting it turned into a model**

**// The first argument of the model i.e ‘Kitten’ is the name of collection that would be created after saving our document using save() method**

const **silence** = new Kitten({ name: 'Hello' }); **// Creating the object of Kitten model named as silence**

console.log(silence.name); // 'Silence'

- A Mongoose **model** is a wrapper on the Mongoose **schema**  OR Model can be termed as a **compiled schema**

- Also , a Model is a class with which we construct documents .

- Each above case each documents will bea kitten with porperties and behaviours as declared in our schema

- A Mongoose schema defines the structure of the document/row , default values , validators , etc whereas a Mongoose model provides an interface to the database for creating , querying , updating , deleting records etc.

//adding methods/functions to **methods property** of a schema and compiling it to model and using it with model objects

kittySchema.methods.speak = function speak(){

var greeting = "helo all myself " + this.name;

}

const Kitten = mongoose.model(**'Kitten'**, kittySchema); // schema compilation to model

const fluffy = new Kitten({ name: 'fluffy' }); // model object creation

const desi = new Kitten({name:’bobo’});

fluffy.**speak();** // using schema-added method **speak()** with model objects

desi.**speak();**

// saving the document to mongodb using **save() method** of mongooose

**fluffy.save();**

**desi.save();**

**silence.save();**

- On saving the document-objects({name:’fluffy’} , {name:’bobo’} , {name:’hello’}) kept in a model-objects(i.e , fluffy , desi , silence) , mongoose creates a collection in the connected database (test) named with the plural form of the first argument passed to our model i.e **‘Kitten’ to ‘Kittens’** , and saves these document-objects in this created collection .

// Displaying/Reading the documents of our **‘Kittens’** collection , through **Kitten** model , using **find() method** of mongoose

const kittens = await **Kitten.find()**;

const filterKittens = await Kitten.find(**{name:”fluffy”}**) // displaying docs using filter objects

const.log(kittens);

const.log(filterKittens);

**Getting post request from a form and saving form data to the database using mongoose**

**Connectin got our test database**

main().catch(err => console.log(err));

async function main() {

await **mongoose.connect('mongodb://localhost:27017/myDatabase');**

}

**Creating the schema of according to the form inputs**

var contactSchema = new **mongoose.Schema**({

name : String,

phone : String,

email : String,

address : String,

desc : String

});

**Creating the model**

var Contact = mongoose.model('Contact',contactSchema);

**Getting data from form and and saving to our database**

app.post('/contact',(req,res)=>{

var myData = new Contact(**req.body**);

**myData.save().**then(()=>{res.send("This document has been saved")}).catch(()=>{res.status(400).send("Error")})

})

- **req.body** contains the form data which is passed inside the model object

- myData.save() returns promise , hence we use then-catch block to prompt messages according to situation

**Deployment**

- **Domain** is the location or address or name of a website . ex:- youtube.com , gmail.com etc

- **Hosting** is the place where the files of our websites are stored . ex:- firebase , heroku , godaddy etc

- **Server** is a physical machine over which we perform the hosting of our website

- **Shared-hosting** : It allows multiple websites to utilize single server

- VPS stands for Virtual Private Server

- Digital Ocean is an online platform which provides VPS sevices or Cloud services

- A droplet in Digital Ocean is a Linux-based VPS/Virtual Machine hosted on their cloud or data-center

- Digital Ocean also provides the feature of **Access Console** which allows to access the terminal of droplet/virtual-server/VPS/VM

- **Putty** is the software which does the job of Access Console

- **SSH(Secure Shell Protocol)** is a cryptographic network protocol , which is used in remote login

- On digital Ocean we will first get the access of a linux VPS and will get into that virtual machine by logging in using provided IP address and password with the help of putty

- On logging in , the VM’s linux terminal will prompt and you can access the virtual system from here only

- we can also get into our cloud server using powershell , by using following commands

> ssh root@IPaddress

> password

- **sudo(superuser do /substitute user do)** keyword gives a general user the access to the server . like , sudo apt update etc

- Apache is a web-server on linux system

- A firewall is a network security device that monitors incoming and outgoing network traffic and permits or blocks data packets based on a set of security rules

- UFW(Uncomplicated firewall) is a frontend for managing firewall rules in linux .

- UWF is used through the command line

- we need to allow incoming traffic over the apache2 web-server with UFW

> sudo ufw app list

Available Applications:

Apache

Apache Full

Open SSH

> sudo ufw allow Apache Full

\*\*Head over Digital Ocean docs for ufw , outgoing and incoming allowances and for setting up the Firewall with UFW\*\*