**Mean Stack Technologies**

A Skill Oriented Course Web Application / Project report

Submitted in the partial fulfillment of the requirements for

the award of the degree of

Bachelor of Technology

In

Computer Science and Engineering

by

**S.Durga Bhavani 20761A05I0**

Under the guidance of

Dr. K. Devi Priya

Assoc. Professor, Dept. of CSE



**Department of Computer Science and Engineering**

**Lakireddy Bali Reddy College of Engineering (Autonomous)**

**Accredited by NAAC & NBA (Under Tier - I)**

**Affiliated to JNTUK, Kakinada; ISO 9001:2015 Certified**

**2022-23**

|  |  |
| --- | --- |
|  | **LAKIREDDY BALI REDDY COLLEGE OF ENGINEERING**  **(AUTONOMOUS)**  **Accredited by NAAC with ‘A’ Grade & NBA (Under Tier - I), ISO 9001:2015 Certified Institution**  **Approved by AICTE, New Delhi and Affiliated to JNTUK, Kakinada**  **L.B. REDDY NAGAR, MYLAVARAM, NTR DIST., A.P.-521 230.**  [hodcse@lbrce.ac.in,](mailto:hodcse@lbrce.ac.in,%20) [cselbreddy@gmail.com,](mailto:cselbreddy@gmail.com) Phone: 08659-222 933, Fax: 08659-222931  **DEPARTMENT OF COMPUTER SCIENCE &ENGINEERING** |

**Certificate**

This is to certify that the Skill Oriented Course entitled “**Mean Stack Technologies**” is being submitted by **S.Durga Bhavani(20761A05I0)** in partial fulfillment for the award of B. Tech in Computer Science & Engineering to the Jawaharlal Nehru Technological University Kakinada is a record of bonafide work carried out by him/her under our guidance.

The results embodied in this Skill Oriented Course Web Application / Project report have not been submitted to any other University or Institute for the award of any degree or diploma.

**Project guide Head of the department**

Dr. K. Devi Priya Dr.D.Veeraiah

Assoc. Professor Professor

**External Examiner**

**ACKNOWLEDGEMENT**

I would like to thank **Dr.K.Devi Priya,** Assoc. Professor**,** CSE department for the encouraging and support in carrying out this Skill Oriented Course Web Application / Project.

I also take the privilege to record my thanks to **Dr. D. Veeraiah,** Professor, Head of the Department of CSE whose encouragement, cooperation and valuable support crown my success.

I express my thanks to the support given by management in completing my Web Application / Project. I also express my sincere gratitude & deep sense of respect to the Principal, **Dr. K. Appa Rao** for making us available all the required assistance and his support and inspiration to carry out this Skill Oriented Course Web Application / Project in the Institute.

I am thankful to the teaching and non-teaching staff of CSE department for their direct as well as indirect help in my Skill Oriented Course Web Application / Project.

I am elated to avail my selves to this opportunity to express my deep sense of gratitude to my parents.

**CONTENTS Page No**

1. Introduction to MEAN Stack Technologies 2-3
2. Java Script 4-13

* Introduction
* Example Programs

1. Node JS 14-20

* Introduction
* Installation of Packages
* FS Module
* Example Programs

1. Express JS 21-24

* Introduction
* How to handle the routes.
* Reading Query Parameters from HTML.
* Example Programs

1. Mongo DB 25-27

* Introduction
* How to establish a Connection
* CRUD Operations

1. Form Creation and Insertion and Accessing application using 28-31

Mongodb and Express JS

1. Web References 32

1.**Introduction to MEAN Stack Technologies :**

MEAN Stack is one of the most popular Technology Stack. It is used to develop a Full Stack Web Application. Although it is a Stack of different technologies, all of these are based on JavaScript language.

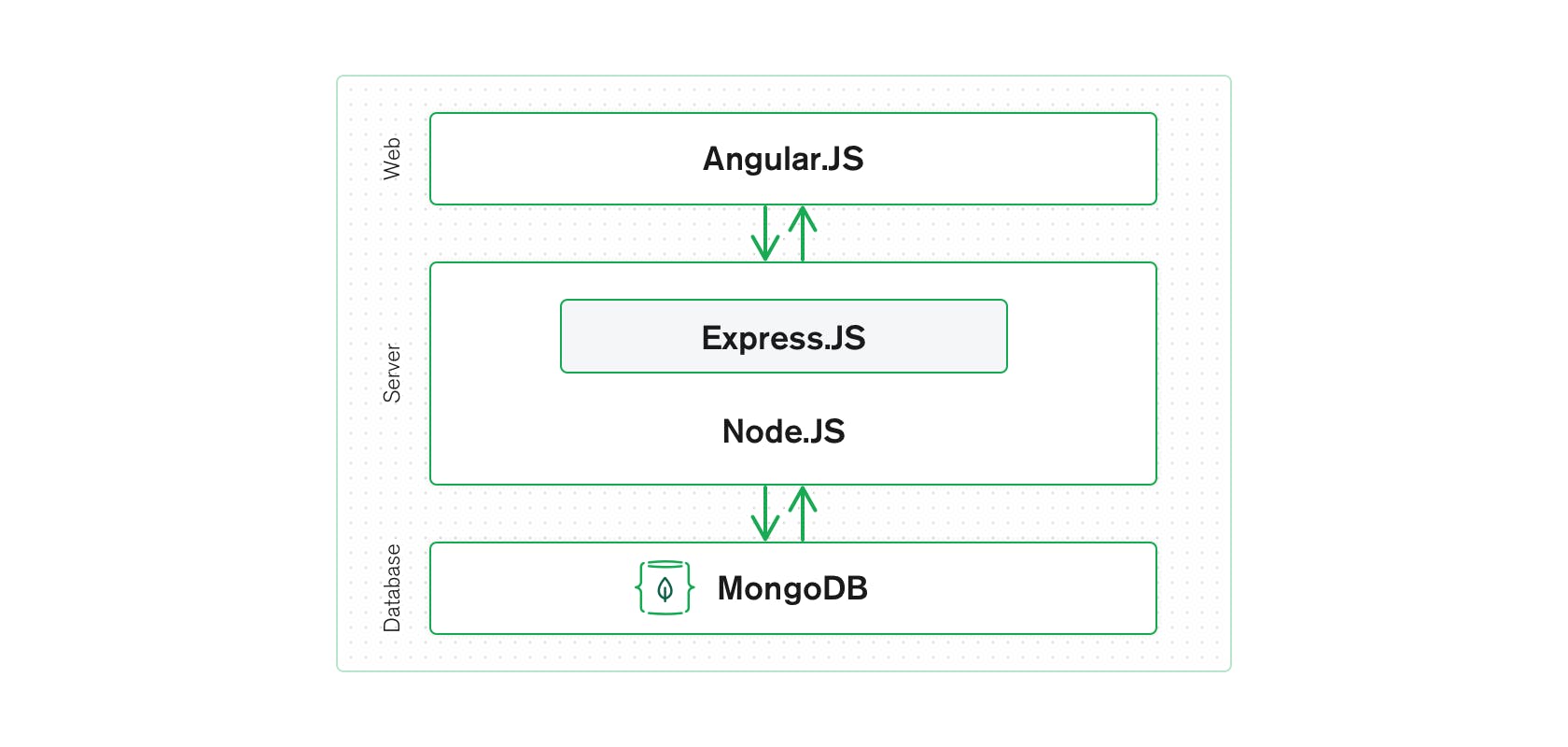
MEAN Stands for:

1. **M** – MongoDB
2. **E** – Express
3. **A** – Angular
4. **N** – Node.js

This stack leads to faster development as well as the deployment of the Web Application. Angular is Frontend Development Framework whereas Node.js, Express, and MongoDB are used for Backend development.

### MEAN stack architecture

The MEAN architecture is designed to make building web applications in JavaScript and handling JSON incredibly easy.



#### Angular.js front end

At the very top of the MEAN stack is Angular.js, the self-styled “JavaScript MVW Framework” (MVW stands for “Model View and Whatever”).

Angular.js allows you to extend your HTML tags with metadata in order to create dynamic, interactive web experiences much more powerfully than, say, building them yourself with static HTML and JavaScript (or jQuery).

Angular has all of the bells and whistles you’d expect from a front-end JavaScript framework, including form validation, localization, and communication with your back-end service.

#### Express.js and Node.js server tier

The next level down is [Express.js](https://expressjs.com/), running on a Node.js server. Express.js calls itself a “fast, unopinionated, minimalist web framework for Node.js,” and that is indeed exactly what it is.

Express.js has powerful models for URL routing (matching an incoming URL with a server function), and handling HTTP requests and responses. By making XML HTTP requests (XHRs), GETs, or POSTs from your Angular.js front end, you can connect to Express.js functions that power your application.

Those functions, in turn, use MongoDB’s Node.js drivers, either via callbacks or using promises, to access and update data in your MongoDB database.

#### MongoDB database tier

If your application stores any data (user profiles, content, comments, uploads, events, etc.), then you’re going to want a database that’s just as easy to work with as Angular, Express, and Node.

That’s where MongoDB comes in: JSON documents created in your Angular.js front end can be sent to the Express.js server, where they can be processed and (assuming they’re valid) stored directly in MongoDB for later retrieval.

Again, if you want to easily get the best of MongoDB, you’ll want to look at [MongoDB Atlas](https://www.mongodb.com/cloud/atlas). This will allow you built-in full database security and cross-cloud scalability with the click of a button. [More on that later on this page.](https://www.mongodb.com/mean-stack#using-mean-stack-with-mongodb-atlas)

**2. Java Script**

**Introduction:**

* JavaScript is the language used for client-side scripting to do client-side validations.
* Also, JavaScript frameworks can be used for writing complex business logic that runs at the client-side.
* **JavaScript** is a lightweight, cross-platform, and interpreted compiled programming language which is also known as the scripting language for webpages.
* It is well-known for the development of web pages, many non-browser environments also use it.

**Features of Java Script:**

* Scripting Language:

JavaScript is a lightweight scripting language made for client-side execution on the browser. Since it is not designed as a general-purpose language and is specially engineered for web applications, the set of libraries is also geared primarily towards web applications.

* Interpreter Based:

JavaScript is an interpreted language instead of a compiled one. In that sense, it is closer to languages like Ruby and Python. The browser interprets JavaScript’s source code, line by line and runs it. In contrast, a compiled language needs to be compiled into a byte-code code executable. Java and C++ are examples of compiled languages.

* Event Handling:

In event is an action or an occurrence in a system that communicates about said occurrence so that you can respond to it somehow. For example, a user clicks on a button, and the system tells you to respond to the button click event with an action, say an information box.

JavaScript enables you to handle events and even generate custom events.

* Light Weight:

JavaScript isn’t a compiled language, so it doesn’t get converted to byte-code beforehand. However, it does follow a paradigm called Just-In-Time (JIT) Compilation. Meaning it gets converted to bytecode just as it’s about to run. This enables JS to be lightweight. Even less powerful devices are capable of running JavaScript.

* Case Sensitive:

JavaScript is highly case sensitive. All keywords, variables, functions names and other identifiers can and must only follow a consistent capitalisation of letters. E.g.:

var hitCounter = 5  
var hitcounter = 5

Here variables hitCounter and hitcounter are both different variables because of the difference in the case. Also, all keywords such as “var” are case sensitive.

* Control Statements:

JavaScript is equipped with control statements like if-else-if, switch-case, and loops like for, while, and do-while loops. These control statements make it a powerful programming language, enabling its user to write complex logic.

**Example Programs:**

1.Implement a Java Script to find the area of a circle.

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Area and Perimeter of a circle</title>

<script>

function circle(radius)

{

this.radius = radius;

// area method

this.area = function ()

{

return Math.PI \* this.radius \* this.radius;

};

// perimeter method

this.perimeter = function ()

{

return 2\*Math.PI\*this.radius;

};

}

var c = new circle(3);

document.writeln('Area =', c.area().toFixed(2));

document.writeln("<br>");

document.writeln('perimeter =', c.perimeter().toFixed(2));

</script>

</head>

<body>

</body>

</html>

**Output:**

Area =28.27  
perimeter =18.85

**2. Write JavaScript code to display the movie details such as movie name, starring,language, and ratings. Initialize the variables with values of appropriate types. Use template literals wherever necessary.**

|  |
| --- |
| <html> |
|  | <head> |
|  | <script> |
|  | var m,star,rate,lan,i; |
|  | for(i=0;i<2;i++) |
|  | { |
|  | document.write("<br>"); |
|  | m=window.prompt("Enter Movie Name:"+m); |
|  | document.write("<br>"); |
|  | star=window.prompt("Enter Starrings"+star); |
|  | document.write("<br>"); |
|  | lan=window.prompt("Enter language:"+lan); |
|  | document.write("<br>"); |
|  | rate=parseInt(window.prompt("Enter rattings:"+rate)); |
|  | document.write("<br>"); |
|  | document.write("moviename:"+m+"<br>"); |
|  | document.write("starring:"+star+"<br>"); |
|  | document.write("language:"+lan+"<br>"); |
|  | document.write("rattings"+rate+"<br>"); |
|  | } |
|  | </script> |
|  | </head> |
|  | </html> |

**OUTPUT:**

Movie name: Bahubali  
starring: Prabhas, Anushka  
language: Telugu, Hindi and English  
rattings:5

movie name: Pushpa  
starring: Allu Arjun, Rashmika  
language: Telugu, Hindi and English  
rattings: 5

**3. Write JavaScript code to book movie tickets online and calculate the total price,considering the number of tickets and price per ticket as Rs. 150. Also, apply a festive season discount of 10% and calculate the discounted amount.**

|  |
| --- |
| <html> |
|  | <body> |
|  | <input id="movies">No.of Seats |
|  | <br><br> |
|  | <button onclick="getPrice()">get total</button> |
|  | <input readonly id="total"> |
|  | <script> |
|  | getPrice=function(){ |
|  | var n=Number(document.getElementById("movies").value); |
|  | const ticket\_cost=150; |
|  | var total\_cost; |
|  | if(n>2 && n<=6){ |
|  | total\_cost=n\*ticket\_cost-(n\*ticket\_cost\*0.05); |
|  | document.getElementById("total").value=total\_cost.toFixed(2); |
|  | //document.write("Amount to be paid"); |
|  | } |
|  | else if(n>=1 && n<=2) { |
|  | total\_cost=n\*ticket\_cost; |
|  | document.getElementById("total").value=total\_cost.toFixed(2); |
|  | //document.write("Amount to be paid"); |
|  | } |
|  | else{ |
|  | alert("Booking not allowed"); |
|  | } |
|  | } |
|  | </script> |
|  |  |
|  |  |
|  | </body> |
|  | </html> |

Output:

No.of Seats  
  
get total 

**4. Creating Arrays, Demonstrating Arrays, Accessing Arrays, Array**

Methods.

<html>

<head>

<script>

const marks=[10,67,36,92,57,89,01];

const names=["cse","it","ece","eee"];

const names1=[1.,2.,3.,4.];

document.write(marks+" ");

document.write("<br>");

document.write("<br>");

document.write("<h2 style='color:green'>");

document.write("After Sorting:");

document.write(marks.sort());

document.write("<br>");

document.write("</h2>");

document.write("<h2 style='color:red'>");

document.write("After joining:");

document.write("<br>");

document.write(names1.join(names[0]));

document.write("<br>");

document.write("<h2 style='color:yellow'>");

document.write("Pushing an element:");

document.write(names.push("ai and ml"));

document.write("<br>");

document.write("<h2 style='color:blue'>");

document.write("Pop an element:");

document.write(marks.pop(92));

document.write(marks);

document.write("<h2 style='color:rayon'>");

document.write("<br>");

document.write("After adding:" );

document.write(names);

document.write("<h2 style='color:black'>");

document.write("<br>");

document.write("After concatanation:");

document.write(names.concat(names1));

</script>

</head>

</html>

**Output:**

10,67,36,92,57,89,1  
After Sorting:1,10,36,57,67,89,92

After joining:  
1cse2cse3cse4

Pushing an element:5

Pop an element:921,10,36,57,67,89

After adding:cse,it,ece,eee,ai and ml

After concatanation:cse,it,ece,eee,ai and ml,1,2,3,4

**5. Introduction to Asynchronous Programming, Callbacks, Promises,Async and Await, Executing Network Requests using Fetch API**

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript setInterval()</h2>

<p>Using setInterval() to display the time every second (3000 milliseconds).</p>

<h1 id="demo"></h1>

<script>

setInterval(myFunction, 3000);

function myFunction() {

let d = new Date();

document.getElementById("demo").innerHTML=

d.getHours() + ":" +

d.getMinutes() + ":" +

d.getSeconds();

}

</script>

</body>

</html>

**Output:**JavaScript setInterval()

Using setInterval() to display the time every second (3000 milliseconds).

# 15:20:38

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript SetTimeout()</h2>

<p>Wait 3 seconds (3000 milliseconds) for this page to change.</p>

<h1 id="demo"></h1>

<script>

function myFunction(value) {

document.getElementById("demo").innerHTML = value;

}

setTimeout(function() { myFunction("Nithin"); }, 3000);

</script>

</body>

</html>

**Output:**

## JavaScript SetTimeout()

Wait 3 seconds (3000 milliseconds) for this page to change.

# Nithin

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript Callbacks</h2>

<p>Do a calculation and then display the result.</p>

<p id="demo"></p>

<script>

function myDisplayer(something) {

document.getElementById("demo").innerHTML = something;

}

function myCalculator(num1, num2, myCallback) {

let sum = num1 + num2;

myCallback(sum);

}

myCalculator(5, 5, myDisplayer);

</script>

</body>

</html>

**Output:**

**JavaScript Callbacks**

Do a calculation and then display the result.

10

**2.Node JS**

**Introduction:**

* Node.js is an open source server environment.
* Node.js allows you to run JavaScript on the server.
* Node.js runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
* It uses asynchronous programming.
* Node.js can generate dynamic page content.
* Node.js can create, open, read, write, delete, and close files on the server.
* Node.js can collect form data.
* Node.js can add, delete, modify data in your database.

**Node.js = Runtime Environment + JavaScript Library**

**Installation of Packages:**

* Node Package Manager (NPM) is a command line tool that installs, updates or uninstalls Node. js packages in your application.
* It is also an online repository for open-source Node. js packages.
* NPM is a collection of all the open-source JavaScript libraries available in this world.
* It is the world’s largest software registry maintained by the Node.js team.
* There are many packages available on NPM which can be used in our application development.
* If we want to globally install any package or tool add **-g** to the command.
* On installing any package globally, that package gets added to the PATH so that we can run it from any location on the computer.

**Syntax:**

**npm install -g <package\_name>**

**Example:**

**npm install –g express**

**FS Module:**

* Node.js comes with the file system (fs) module fs to perform file operations.

What is fs module?

* The fs module contains the standard file operations for manipulating files and dealing with the computing platform's file system.
* To use ‘fs’ module we need to import them.

const fs = require('fs');

* Some of the file operations are:

1. Writing data to a file
2. Reading data from a file
3. Updating content in a file

**Example Programs:**

|  |
| --- |
|  |
|  | CODE: |
|  | var fs = require("fs"); |
|  |  |
|  | // Asynchronous read |
|  | fs.readFile('input.txt', function (err, data) { |
|  | if (err) { |
|  | return console.error(err); |
|  | } |
|  | console.log("Asynchronous read: " + data.toString()); |
|  | }); |
|  |  |
|  | **OUTPUT:** |
|  | Asynchronous read: My Name: This is my name |
|  |  |
|  | Synchronous approach: |
|  | CODE: |
|  | var fs = require("fs"); |
|  |  |
|  | // Synchronous read |
|  | var data = fs.readFileSync('input.txt'); |
|  | console.log("Synchronous read: " + data.toString()); |
|  |  |
|  | **OUTPUT:** |
|  | Synchronous read: Sai Nithin: This is my name |

**2.opening a file:**

|  |
| --- |
| CODE: |
|  | var fs = require("fs"); |
|  |  |
|  | // Asynchronous - Opening File |
|  | console.log("opening file!"); |
|  | fs.open('input.txt', 'r+', function(err, fd) { |
|  | if (err) { |
|  | return console.error(err); |
|  | } |
|  | console.log("File open successfully"); |
|  | }); |
|  | OUTPUT: |
|  | opening file! |
|  | File open successfully |

**3.Reading a file:**

|  |
| --- |
| CODE: |
|  | var fs = require("fs"); |
|  | var buf = new Buffer(1024); |
|  |  |
|  | console.log("opening an existing file"); |
|  | fs.open('input.txt', 'r+', function(err, fd) { |
|  | if (err) { |
|  | return console.error(err); |
|  | } |
|  | console.log("File opened successfully!"); |
|  | console.log("reading the file"); |
|  |  |
|  | fs.read(fd, buf, 0, buf.length, 0, function(err, bytes){ |
|  | if (err){ |
|  | console.log(err); |
|  | } |
|  | console.log(bytes + " bytes read"); |
|  |  |
|  | // Print only read bytes to avoid junk. |
|  | if(bytes > 0){ |
|  | console.log(buf.slice(0, bytes).toString()); |
|  | } |
|  | }); |
|  | }); |
|  | OUTPUT: |
|  | opening an existing file |
|  | File opened successfully! |
|  | reading the file |
|  | 40 bytes read |
|  | My Name: This is my name |

**4.Writing a file:**

|  |
| --- |
| CODE: |
|  | var fs = require("fs"); |
|  |  |
|  | console.log("writing into existing file"); |
|  | fs.writeFile('input.txt', 'Nithin', function(err) { |
|  | if (err) { |
|  | return console.error(err); |
|  | } |
|  |  |
|  | console.log("Data written successfully!"); |
|  | console.log("Let's read newly written data"); |
|  |  |
|  | fs.readFile('input.txt', function (err, data) { |
|  | if (err) { |
|  | return console.error(err); |
|  | } |
|  | console.log("Asynchronous read: " + data.toString()); |
|  | }); |
|  | }); |
|  | OUTPUT: |
|  | writing into existing file |
|  | Data written successfully! |
|  | Let's read newly written data |
|  | Asynchronous read: My Name |

**5.Appending a file:**

|  |
| --- |
| CODE: |
|  | var fs = require('fs'); |
|  |  |
|  | var data = "\nMy Name"; |
|  |  |
|  | // Append data to file |
|  | fs.appendFile('input.txt', data, 'utf8', |
|  |  |
|  | // Callback function |
|  | function(err) { |
|  | if (err) throw err; |
|  |  |
|  | // If no error |
|  | console.log("Data is appended to file successfully.") |
|  | }); |
|  | OUTPUT: |
|  | Data is appended to file successfully. |
|  | Before Appending Data to input.txt file: |
|  | My Name: This is my name |
|  | After Appending Data to input.txt file:My Name |
|  |  |

**6.Deleting file:**

|  |
| --- |
| CODE: |
|  | var fs = require("fs"); |
|  |  |
|  | console.log("deleting an existing file"); |
|  | fs.unlink('input.txt', function(err) { |
|  | if (err) { |
|  | return console.error(err); |
|  | } |
|  | console.log("File deleted successfully!"); |
|  | }); |
|  | OUTPUT: |
|  | deleting an existing file |
|  | File deleted successfully! |

**7.Crypto Module:**

demonastrating ENCRYPTION and DECRYPTION:

ENCRYPT the text "abc":

CODE:

var crypto = require('crypto');

var mykey = crypto.createCipher('aes-128-cbc', 'mypassword');

var mystr = mykey.update('abc', 'utf8', 'hex');

mystr += mykey.final('hex');

console.log(mystr);

OUTPUT:

34feb914c099df25794bf9ccb85bea72

DECRYPT the text "abc":

CODE:

var crypto = require('crypto');

var mykey = crypto.createDecipher('aes-128-cbc', 'mypassword');

var mystr = mykey.update('34feb914c099df25794bf9ccb85bea72', 'hex', 'utf8');

mystr += mykey.final('utf8');

console.log(mystr);

OUTPUT:

abc

**Server module:**

|  |
| --- |
| const http = require("http"); |
|  | var server = http.createServer((req, res) => { |
|  | res.write("Hello World! I have created my first server! by using node.js"); |
|  | res.end(); |
|  | }); |
|  | server.listen(3000); |
|  | console.log("Server started... Running on localhost:3000"); |

**}**

OUTPUT:

Hello World! I have created my first server! by using node.js

**4.Express JS**

**Introduction:**

* Express.js is a web framework for Node.js. It is a fast, robust and asynchronous in nature.
* Express is a layer built on Node.js which helps us to manage our web server and routes.
* It provides a set of features to develop web and mobile applications.
* Install express from the node terminal using the command:

npm install –g express

**How to handle the routes:**

* Routing refers to determining how an application responds to a client request to a particular endpoint, which is a URI (or path) and a specific HTTP request method (GET, POST, and so on).
* Each route can have one or more handler functions, which are executed when the route is matched.
* Syntax:

app.METHOD(PATH, HANDLER);

app--- Instance of express module.

METHOD--- is a type of HTTP REQUEST( get,post,put,delete)

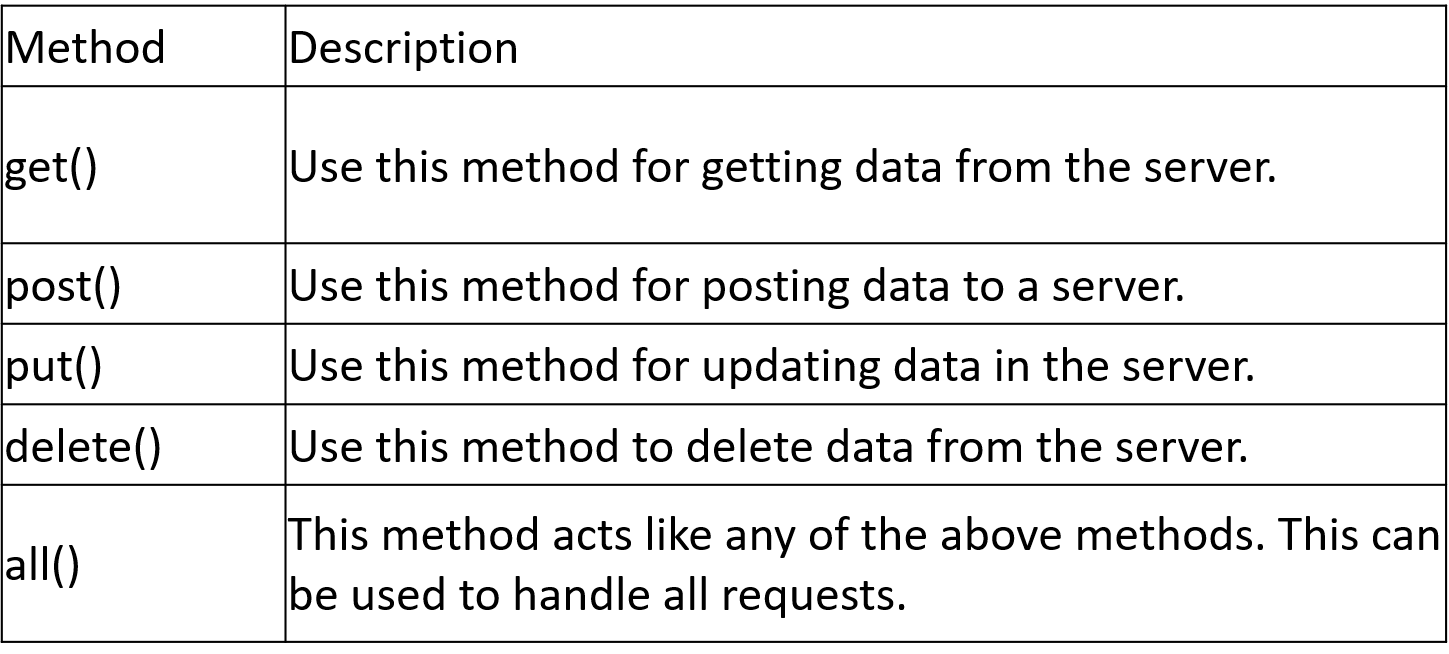
PATH—Path on the server

Handler– Function to be executed when the route is matched.

The application object has different methods corresponding to each of the HTTP verbs (GET, POST, PUT, DELETE).

These methods are used to receive HTTP requests.

Below are the commonly used route methods and their description:



* An Express.js Route defines how our Express application can respond to a client request to with a specific URI (or path) and a specific HTTP request method (GET, POST, etc.).

**Example:**

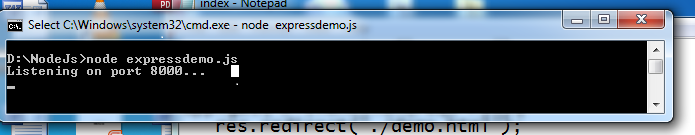
var express = require('express')

var app = express();

var server = app.listen(8000, function(){

console.log('Listening on port 8000...');

});



**Reading query Parameters from HTML:**

* The HTTP response object has information about the response sent from the server to the client. The response object is created along with the request object and is commonly represented by a variable named res.
* res.send(‘Message');
* res.download();--- To download a file
* res.json()--- to send response in JSON Format
* res.redirect()– to redirect response to another URL.
* For example, in the URL, *'http://localhost:3000/user/Smith'*, if we want to capture the portion of URL which contains "Smith", then
* app.get(“/user/:uname”);
* To retrieve the parameters: **req.params.uname**
* **We can retrieve multiple parameters also** *'http://localhost:3000/user/Smith/T831’*
* ***app.get(“/user/:uname/:id”);***

**Example Programs:**

var express = require('express')

var app = express()

// route that gets executed for GET request and the request url path '/' or root

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

res.send('This is a Sample Example for Express.js .')

})

// route that gets executed for GET request and the request url path '/hello/'

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

res.send('Welcome to node express.js.')

})

// route that gets executed for GET request and the request url path '/bye/'

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

res.send('hello page.')

})

// route that gets executed for GET request and the request url path '/bye/'

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

res.send('bye page.')

})

// route that gets executed for GET request and the request url path '/bye/'

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

res.send('Thank you page.')

})

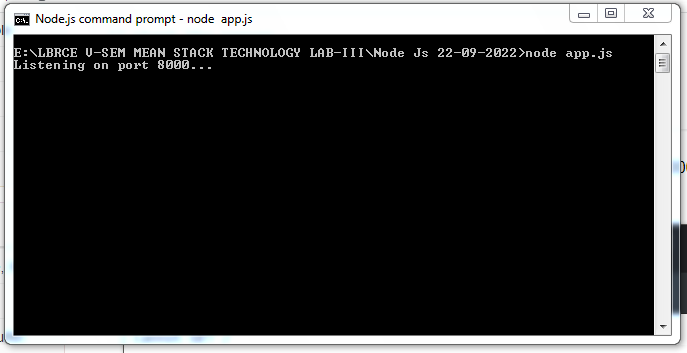
// start the server

var server = app.listen(8000, function(){

console.log('Listening on port 8000...')

})

**Output:**



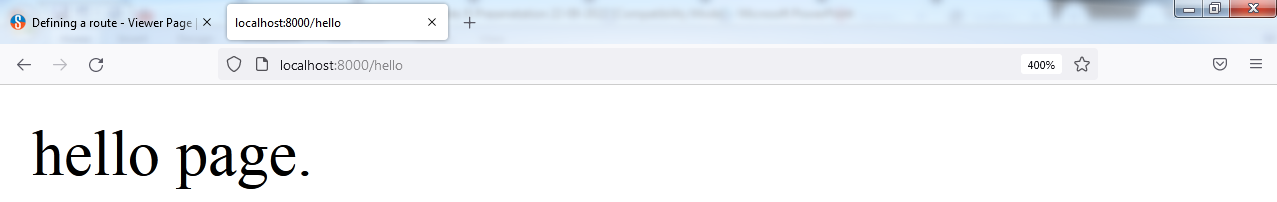
**GET request with URL path http://localhost:8000**



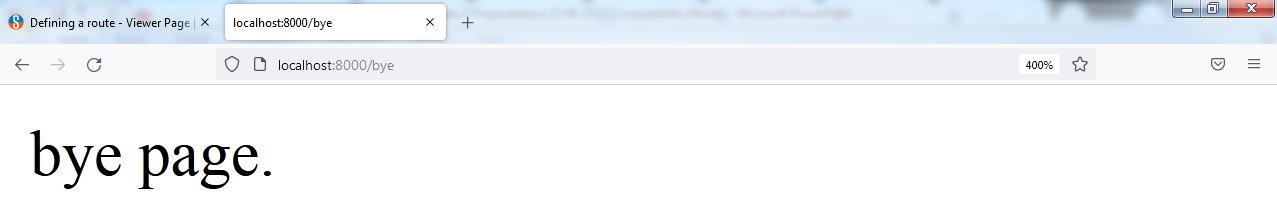
**GET request with URL path http://localhost:8000/welcome**



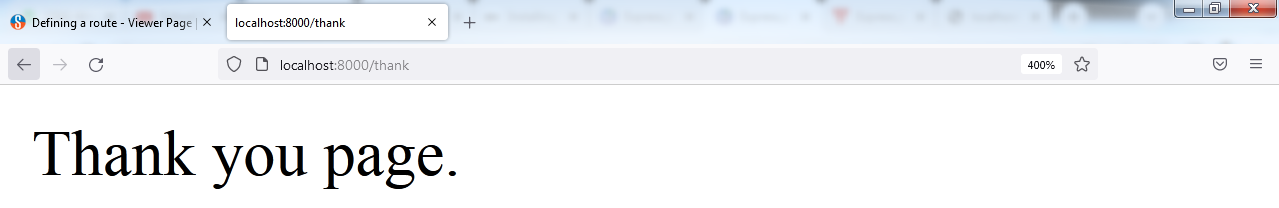
**GET request with URL path http://localhost:8000/hello**



**GET request with URL path http://localhost:8000/bye**



**GET request with URL path http://localhost:8000/thank**



**5. Mongo DB**

**Introduction:**

MongoDB stores documents in [collections](https://www.mongodb.com/docs/manual/core/databases-and-collections/#std-label-collections). Collections are analogous to tables in relational databases.

* Read-only [Views](https://www.mongodb.com/docs/manual/core/views/) (Starting in MongoDB 3.4)
* [On-Demand Materialized Views](https://www.mongodb.com/docs/manual/core/materialized-views/) (Starting in MongoDB 4.2).
* High Performance
* MongoDB provides high performance data persistence. In particular,
* Support for embedded data models reduces I/O activity on database system.
* Indexes support faster queries and can include keys from embedded documents and arrays.
* The MongoDB Query API supports [read and write operations (CRUD)](https://www.mongodb.com/docs/manual/crud/) as well as:
* [Data Aggregation](https://www.mongodb.com/docs/manual/core/aggregation-pipeline/)
* [Text Search](https://www.mongodb.com/docs/manual/text-search/) and [Geospatial Queries.](https://www.mongodb.com/docs/manual/tutorial/geospatial-tutorial/)

**How to establish a Connection:**

**const mongoose = require('mongoose'); // importing mongoose library**

**mongoose.connect('mongodb://localhost:27017/Database\_Name');**

* Then Create the Schema

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

**const schema = new mongoose.Schema({ property\_1: Number, property\_2: String });**

* If you want any field to be mandatory, use the required property.

**const schema = mongoose.Schema({**

**name: {**

**required: true,**

**},**

**});**

You can declare a schema type using the type directly, or an object with a type property.

**const schema = mongoose.Schema({**

**property\_1: {**

**type: String,**

**},**

**property\_2: {**

**type: Number,**

**},**

**});**

* To use our schema definition, we need to wrap the **Schema** into a **Model** object we can work with.
* The model provides an object which provides access to query documents in a named collection.
* Schemas are compiled into models using the **model()** method.

**const Model = mongoose.model(name , schema) ;**

The first argument is the singular name of the collection for which you are creating a Model.

**Example:**

**const NotesModel = mongoose.model("mynotes", myNotesSchema);**

**CRUD Operations:**

const express = require('express');

const app = express();

const mongoose = require('mongoose');

mongoose.connect("mongodb://localhost:27017/CSE\_C",(err)=>{

if(err)

console.log("DB not connected-Error");

else

console.log("DB Connected");

});

const ns= new mongoose.Schema({

name:String,

age:Number,

phno:Number

});

const nm = new mongoose.model("records",ns);

//const data = new nm({name:'Nithin',age:20,phno:1234567890});

//const d1 = new nm({name:'Rajesh',age:20,phno:0123456789});

//data.save();

//d1.save();

nm.insertMany([{name:'Raju',age:32,phno:8587887434},

{name:'Rani',age:40,phno:8587834443},

{name:'Ram',age:32,phno:8587887422}

]).then(function(){

console.log("Data inserted")

}).catch(function(error){

console.log(error)

});

**6.Form Creation and Insertion and Accessing application using**

**Mongodb and Express JS**

**Index.html**

<!DOCTYPE html>  
<html lang = "en">  
<head>  
  <meta charset = "UTF-8">  
  <link rel = "stylesheet" href="style.css">  
  <title> My Form </title>  
</head>  
<body>  
  <header>  
    <h1>Customer Database</h1>  
  </header>  
  <form action="/add" method="POST">  
    <fieldset>  
      <h3>Add new customer</h3>  
      <label>Customer ID</label>  
      <input type ="text" id = 'id' name="id" required>  
      <br><br>  
      <label>Name</label>  
      <input type="text" id = "name" name="name" required>  
      <br><br>  
      <button type ="reset">Reset</button>  
      <button type ="submit">Submit</button>  
    </fieldset>  
  </form>  
<form action="/view" method="POST">  
  <fieldset>  
    <h3>View an Customer</h3>  
    <label>Customer ID</label>  
    <input type="text" id="id" name="id" required>  
    <br><br>  
    <button type ="reset">Reset</button>  
    <button type ="submit">Submit</button>  
    <br><br><br>  
  </fieldset>  
 </form>  
<form action="/update" method="POST">  
  <fieldset>  
    <h3>Update an Customer</h3>  
    <label>Customer ID</label>  
    <input type ="text" id = 'id' name="id" required>  
    <br><br>  
    <label>New Name</label>  
    <input type="text" id = "name" name="name" required>  
    <br><br>  
    <button type ="reset">Reset</button>  
    <button type ="submit">Submit</button>  
  </fieldset>  
 </form>  
<form action="/delete" method="POST">  
  <fieldset>  
    <h3>Delete an Customer</h3>  
    <label>Customer ID</label>  
    <input type ="text" id = 'id' name="id" required>  
    <br><br>  
    <button type ="reset">Reset</button>  
    <button type ="submit">Submit</button>  
    <br><br><br>  
  </fieldset>  
 </form>  
  
  </body>  
</html>

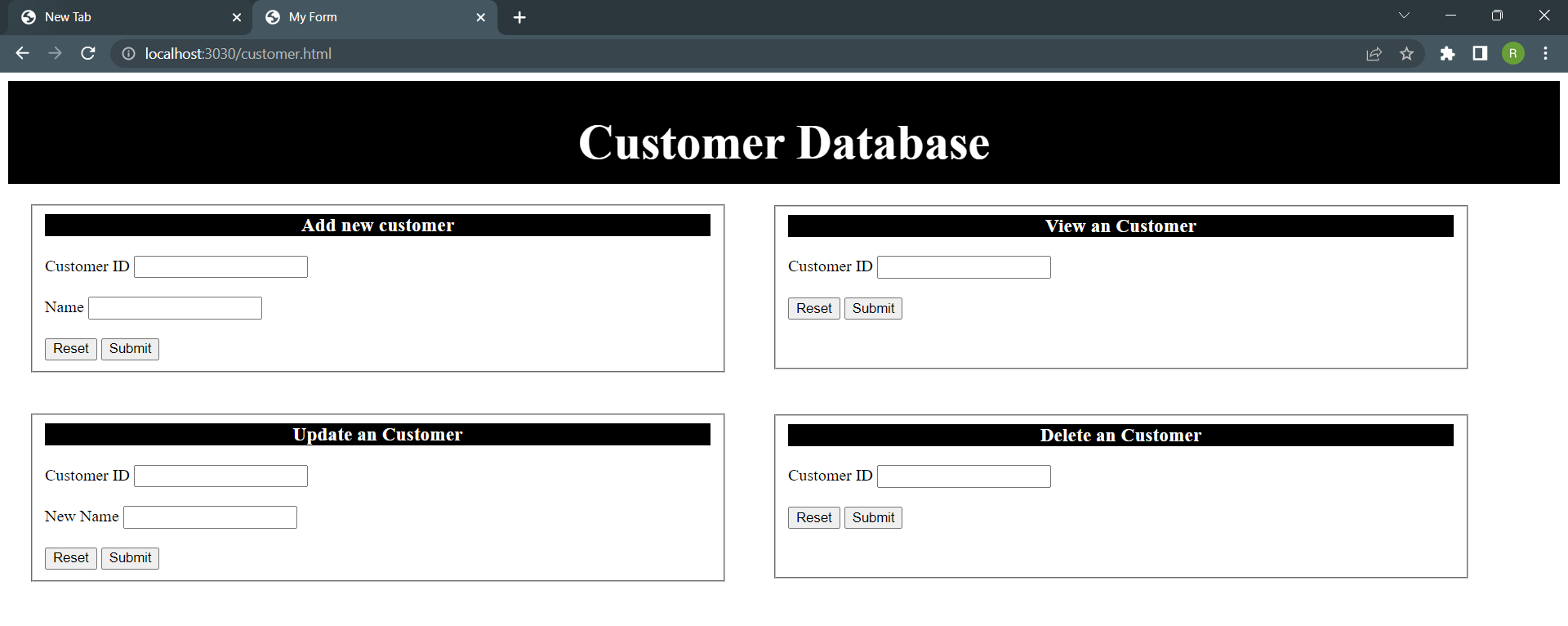
  body{  
 background-color: #ffffff;  
}  
header{  
 text-align: center;  
 height: 100px;  
 padding-top: 1px;  
 background-color: #000000;  
 color: #FFFFFF;  
 font-size: x-large;  
}  
h3{  
 margin-top: 2px;  
 text-align: center;  
 background-color: #000000;  
 color: #FFFFFF;  
}  
form{  
 width: 45%;  
 display: inline-block;  
 margin: 20px;  
}

**Index.js**

var express = require("express")  
var bodyParser = require("body-parser")  
var mongoose = require("mongoose")  
const app = express()  
app.use(bodyParser.json())  
app.use(express.static('public'))  
app.use(bodyParser.urlencoded({  
    extended:true  
}))  
mongoose.connect("mongodb://[127.0.0.1:27017/users](http://127.0.0.1:27017/users)",(err)=>{  
if(err)  
console.log("DB not connected");  
else  
console.log("DB Connected");  
});  
const ns=new mongoose.Schema({  
c\_id:String,  
name:String});  
const nm=new mongoose.model("records",ns);  
var db = mongoose.connection;  
db.on('error',()=>console.log("Error in Connecting to Database"));  
db.once('open',()=>console.log("Connected to Database"))  
  
[app.post](http://app.post/)("/add",(req,res)=>{  
    var id = [req.body.id](http://req.body.id/);  
    var name = [req.body.name](http://req.body.name/);  
    var data =new nm( {  
        "c\_id": id,  
        "name" : name  
    });  
data.save();  
        console.log("Record Inserted Successfully");  
res.send("Record inserted");  
})  
[app.post](http://app.post/)("/delete",(req,res)=>{  
    var id = [req.body.id](http://req.body.id/);  
    nm.deleteOne({"c\_id":id},function(err,data){  
if(err)  
{  
console.log(err);  
}

else  
{  
console.log("Deleted record");  
}  
});  
res.send("record deleted");  
console.log("Record Deleted Successfully");  
})  
[app.post](http://app.post/)("/update",(req,res)=>{  
    const old=[req.body.id](http://req.body.id/);  
const new2=[req.body.name](http://req.body.name/);  
const new1={$set:{"name":new2}};  
nm.findOneAndUpdate({"c\_id":old},new1,function(err){  
  if(err){  
    console.log(err);  
    }  
    else{  
        console.log("updated");  
  }  
});  
res.send("record updated");  
console.log("Record updated successfully");  
})  
[app.post](http://app.post/)("/view",(req,res)=>{  
    var id=[req.body.id](http://req.body.id/);  
    nm.find({"c\_id":id},function(err,res){  
    if(err)  
{  
console.log(err);  
}  
else  
{  
//res.send('<p>Customer\_ID: ' + req.body['id']+'</p><p>Name: '+req.query[id]+'</p>');  
console.log(res);  
}  
});  
res.send("view your details in the console");  
console.log("VIEW DETAILS");  
})  
app.get("/",(req,res)=>{  
    res.set({  
        "Allow-access-Allow-Origin": '\*'  
    })  
    return res.redirect('customer.html');  
}).listen(3030);  
  
console.log("listening on port 3030");

OUTPUT:-

****

**7.Web References**

[**https://www.javatpoint.com/mean-stack-tutorial**](https://www.javatpoint.com/mean-stack-tutorial)

[**https://www.emizentech.com/blog/mean-stack.html**](https://www.emizentech.com/blog/mean-stack.html)

[**https://www.w3schools.com/js/**](https://www.w3schools.com/js/)

[**https://nodejs.org/en/**](https://nodejs.org/en/)

[**https://expressjs.com/**](https://expressjs.com/)

[**https://mongodb.com/mean-stack/**](https://mongodb.com/mean-stack/)