SVKM’s NMIMS University

Mukesh Patel School of Technology Management & Engineering

**MBA Tech SEMESTER V**

**SUBJECT: Advanced Web Programming Practical Assignment: 8**

**Part A (To be referred by students)**

**Topic: Node JS (modules and GUI application)**

**Aim:**

Implement the concepts of modules- how to create module, file system module and create a web application using Express JS module.

**Prerequisite:** HTML, CSS, and JavaScript.

.

**Outcome: After successful completion of this practical students will be able to**

1. Understand how to create server side application.
2. Will be able to understand run NPM and install different modules using the same.

**Theory:**

* Node.js is a very powerful JavaScript-based platform built on Google Chrome's JavaScript V8 Engine.
* It is used to develop I/O intensive web applications like video streaming sites, single-page applications, and other web applications
* Node.js is an open-source, cross-platform runtime environment used for the development of server-side web applications.
* Node.js is based on an event-driven architecture and a non-blocking Input/Output API that is designed to optimize an application's throughput and scalability for real-time web applications

**How to Download & Install Node.js - NPM on Windows**

* The first step is the installation of the node.js framework.
* The Node.js framework is available for a variety of operating systems right from Windows to Ubuntu and OS X.
* Node.js also has the ability to embedded external functionality or extended functionality by making use of custom modules.
* These modules have to be installed separately.
* An example of a module is the MongoDB module which allows you to work with MongoDB databases from your Node.js application.

**The first steps in using Node.js is the installation of the Node.js libraries on the client system.**

* Go to the site https://nodejs.org/en/download/ and download the necessary binary files.

**Running your first Hello World application in Node.js**

* Create file Node.js with file name firstprogram.js

var http = require('http');

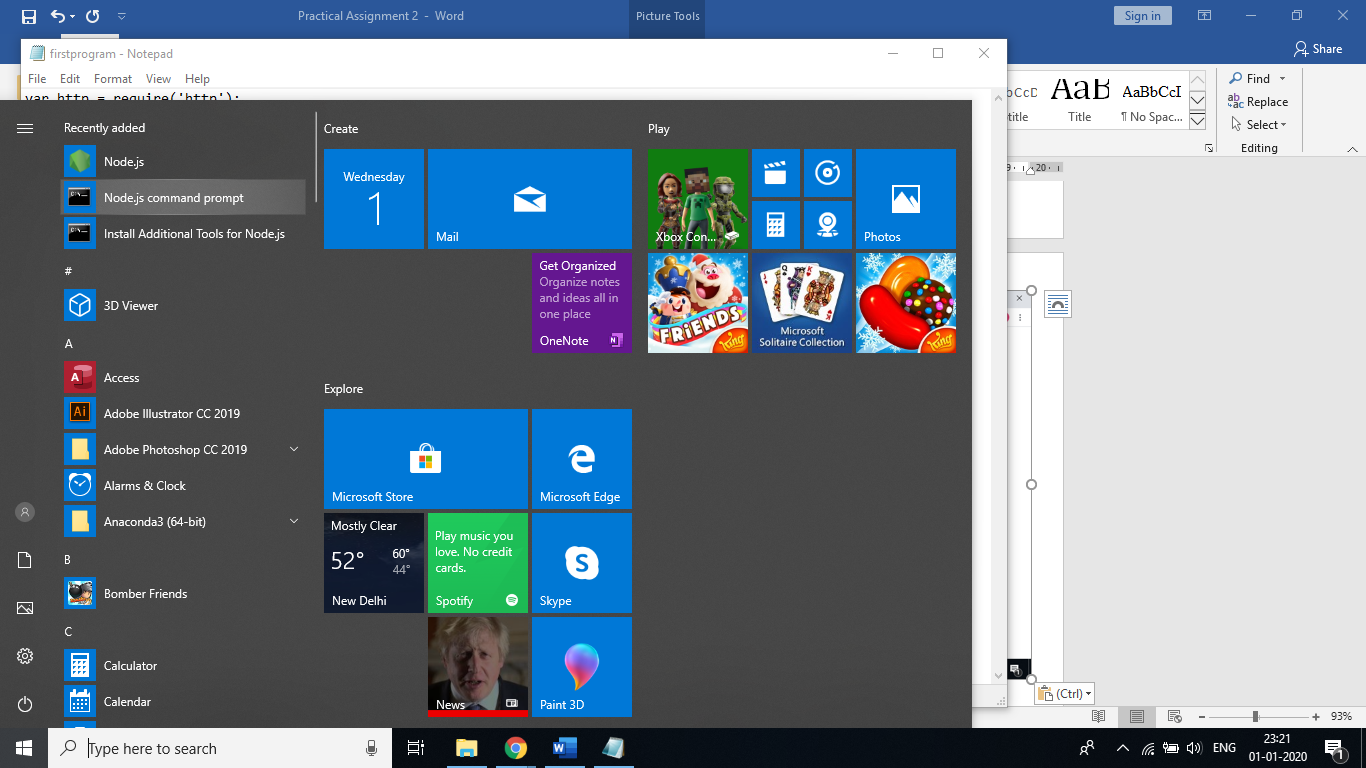
http.createServer(function (req, res) {

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

res.end('Hello World!');

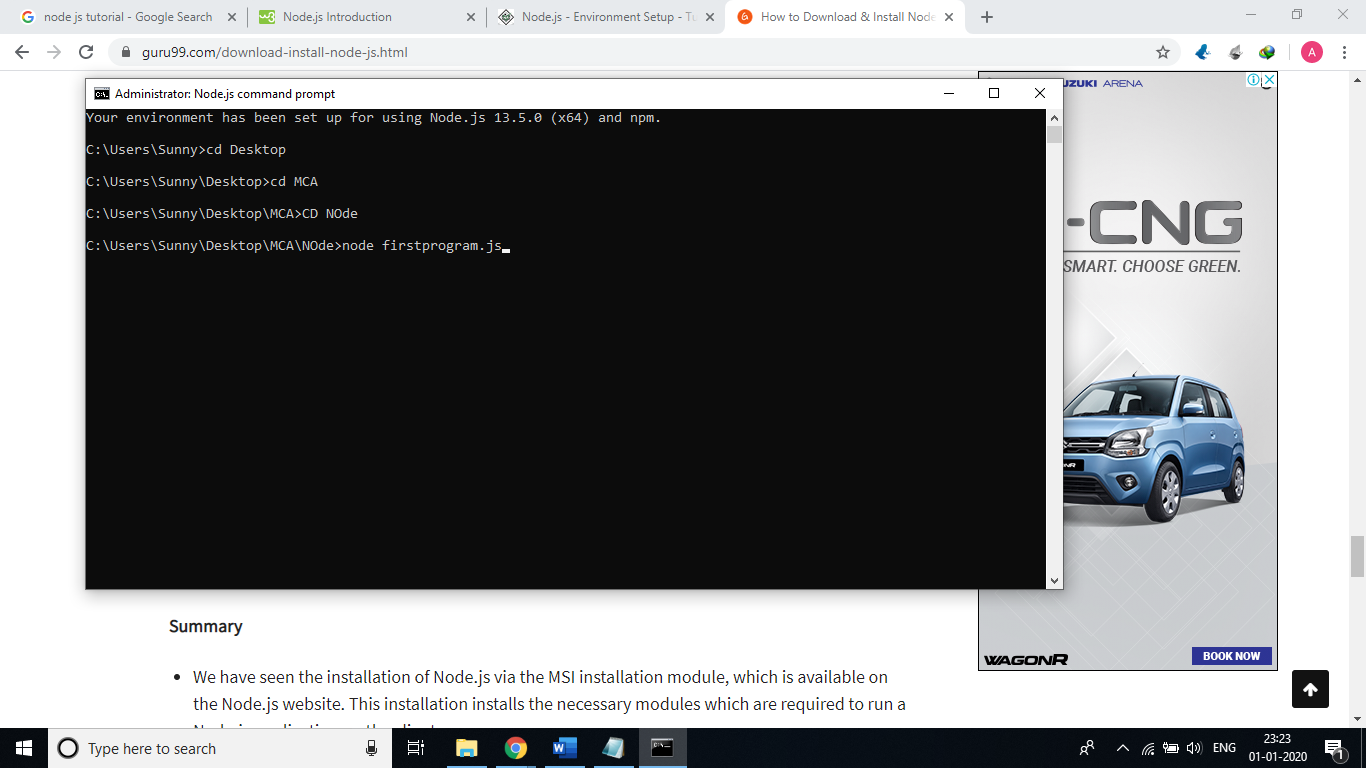
}).listen(8080);

* Go to Node JS command prompt.

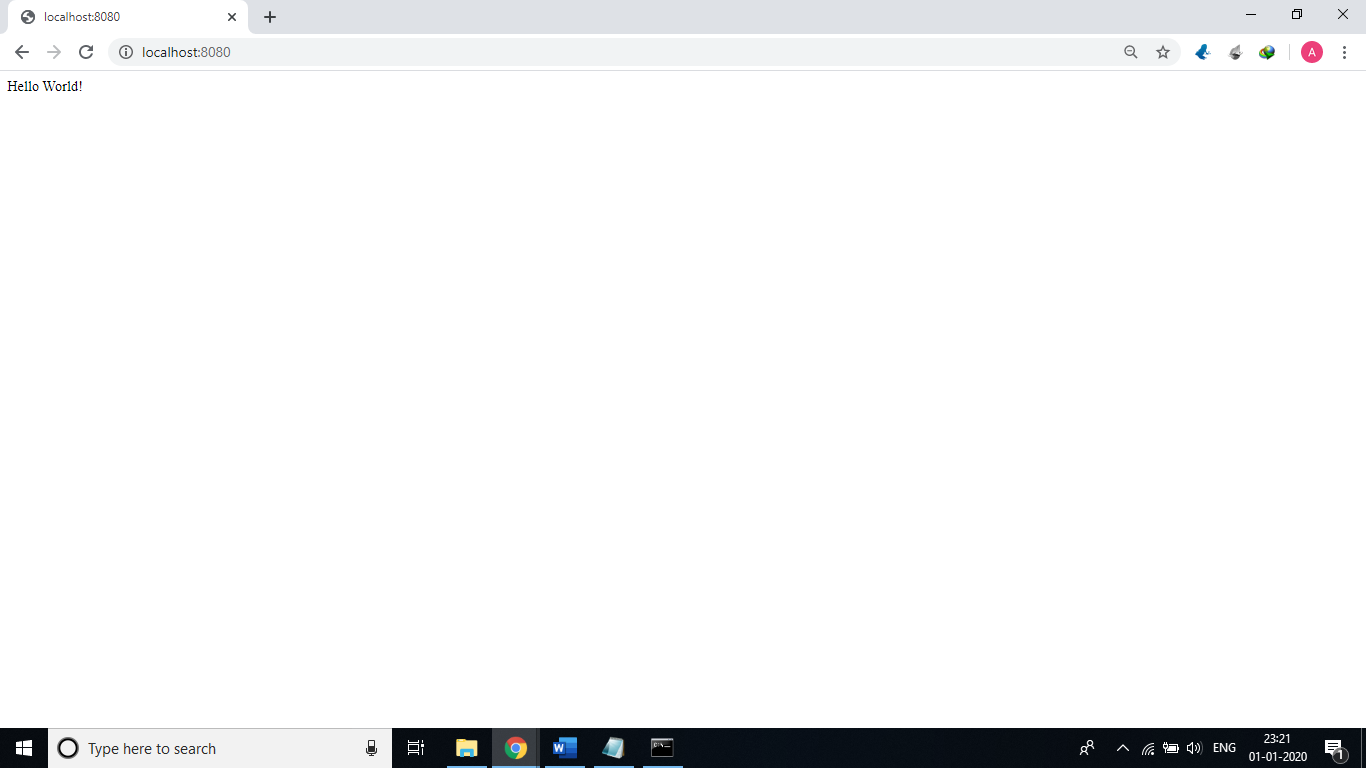


**no**

* In the command prompt, navigate to the folder where the file is stored. Enter the command Node firstprogram.js



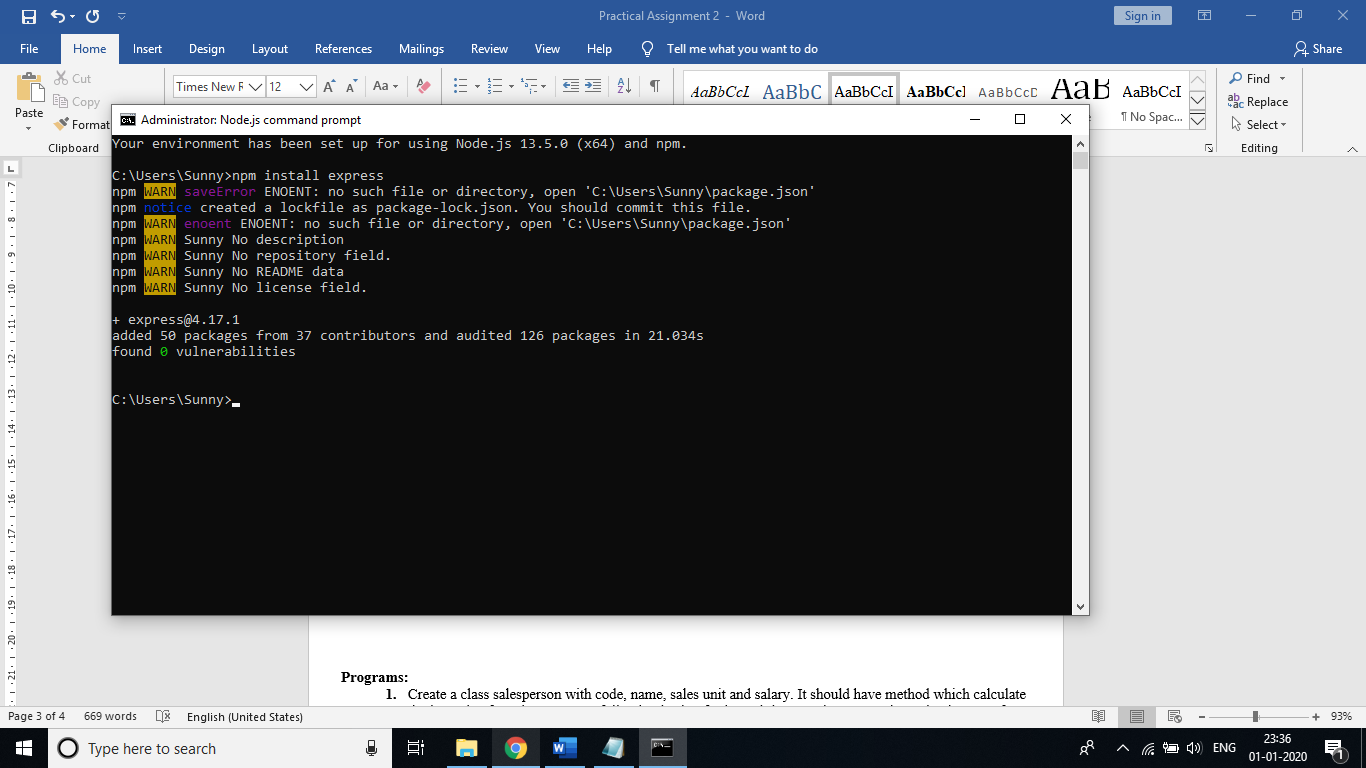
* Now, your computer works as a server! If anyone tries to access your computer on port 8080, they will get a "Hello World!" message in return!
* Start your internet browser, and type in the address: http://localhost:8080



**Using modules in Node.js**

* In order to use modules in a Node.js application, they first need to be installed using the Node package manager.
* On Node Package Manager you have two options: bypass or set a certificate file.
* npm config set strict-ssl false

**Example : npm install express**



* The above command will download the necessary files which contain the "express modules" and take care of the installation as well
* Once the module has been installed, in order to use a module in a Node.js application, you need to use the 'require' keyword. This keyword is a way that Node.js uses to incorporate the functionality of a module in an application.

var express=require('express');

var app=express();

app.set('view engine','jade');

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

{

});

var server=app.listen(3000,function()

{

});

1. In the first statement itself, we are using the "require" keyword to include the express module. The "express" module is an optimized JavaScript library for Node.js development. This is one of the most commonly used Node.js modules.
2. After the module is included, in order to use the functionality within the module, an object needs to be created. Here an object of the express module is created.
3. Once the module is included using the "require" command and an "object" is created, the required methods of the express module can be invoked. Here we are using the set command to set the view engine, which is used to set the templating engine used in Node.js.
4. Here we are using the listen to method to make the application listen on a particular port number.

**Creating NPM modules**

* Node.js has the ability to create custom modules and allows you to include those custom modules in your Node.js application.

**Let's look at a simple example of how we can create our own module and include that module in our main application file. Our module will just do a simple task of adding two numbers.**

Step 1) Create a file called "Addition.js" and include the below code. This file will contain the logic for your module.

* Below is the code which would go into this file;

var exports=module.exports={};

exports.AddNumber=function(a,b)

{

return a+b;

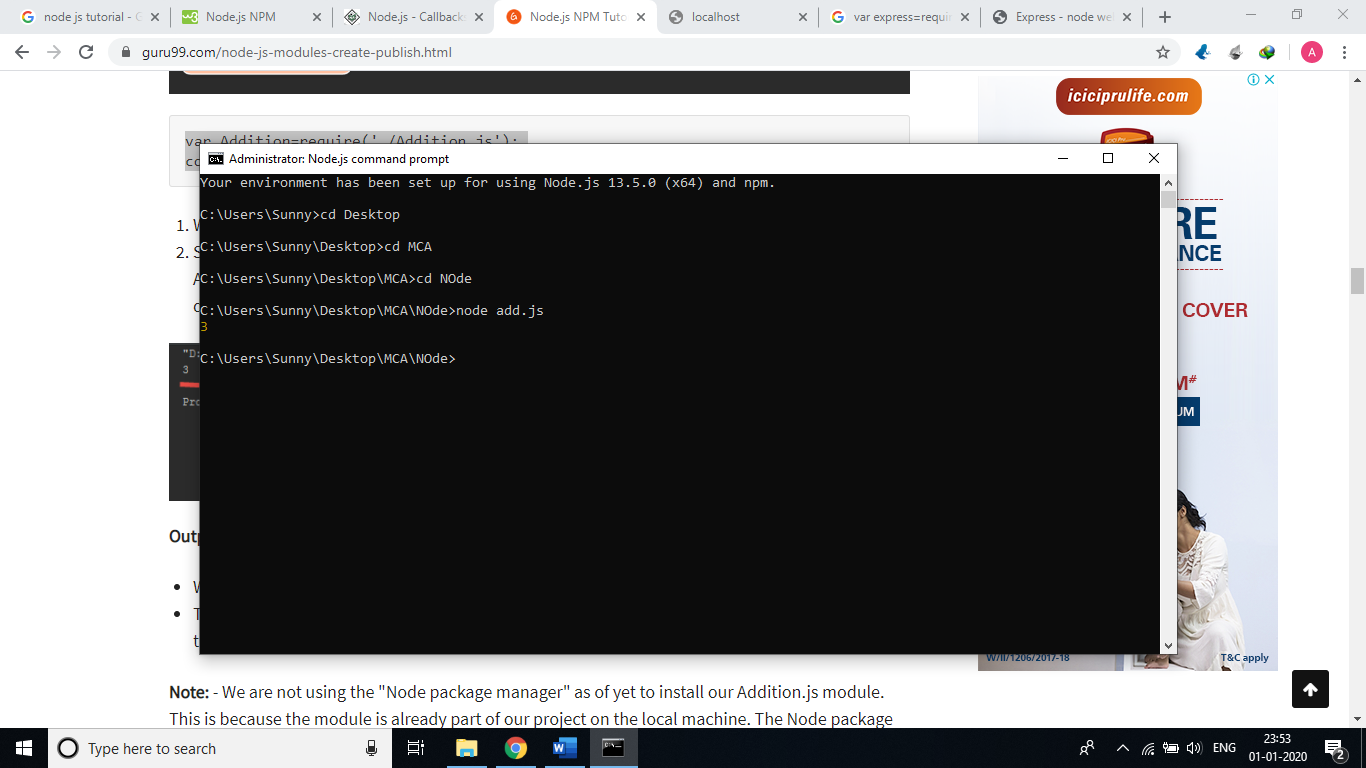
};

1. The "exports" keyword is used to ensure that the functionality defined in this file can actually be accessed by other files.
2. We are then defining a function called 'AddNumber'. This function is defined to take 2 parameters, a and b. The function is added to the module "exports" to make the function as a public function that can be accessed by other application modules.
3. We are finally making our function return the added value of the parameters.

Step 2) Create a file called "app.js,"

var Addition=require('./Addition.js');

console.log(Addition.AddNumber(1,2));



FS Module

The fs module provides a lot of very useful functionality to access and interact with the file system.

There is no need to install it. Being part of the Node.js core, it can be used by simply requiring it:

const fs = require('fs')

Methods

* fs.access(): check if the file exists and Node.js can access it with its permissions
* fs.appendFile(): append data to a file. If the file does not exist, it's created
* fs.chmod(): change the permissions of a file specified by the filename passed. Related: fs.lchmod(), fs.fchmod()
* fs.chown(): change the owner and group of a file specified by the filename passed. Related: fs.fchown(), fs.lchown()
* fs.close(): close a file descriptor
* fs.copyFile(): copies a file
* fs.createReadStream(): create a readable file stream
* fs.createWriteStream(): create a writable file stream
* fs.link(): create a new hard link to a file
* fs.mkdir(): create a new folder
* fs.mkdtemp(): create a temporary directory
* fs.open(): set the file mode
* fs.readdir(): read the contents of a directory
* fs.readFile(): read the content of a file. Related: fs.read()
* fs.readlink(): read the value of a symbolic link
* fs.realpath(): resolve relative file path pointers (., ..) to the full path
* fs.rename(): rename a file or folder
* fs.rmdir(): remove a folder
* fs.stat(): returns the status of the file identified by the filename passed. Related: fs.fstat(), fs.lstat()
* fs.symlink(): create a new symbolic link to a file
* fs.truncate(): truncate to the specified length the file identified by the filename passed. Related: fs.ftruncate()
* fs.unlink(): remove a file or a symbolic link
* fs.unwatchFile(): stop watching for changes on a file
* fs.utimes(): change the timestamp of the file identified by the filename passed. Related: fs.futimes()
* fs.watchFile(): start watching for changes on a file. Related: fs.watch()
* fs.writeFile(): write data to a file. Related: fs.write()

**Example**

**Reading a file**

const fs = require('fs')

fs.readFile('test.txt', 'utf8' , (err, data) => {

if (err) {

console.error(err)

return

} console.log(data)})

**Writing a file**

const fs = require('fs')

const content = 'Some content!'

fs.writeFile('test.txt', content, err => {

if (err) {

console.error(err)

return

}

//file written successfully

})

**Express JS**

* Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.
* Express provides methods to specify what function is called for a particular HTTP verb (GET, POST, SET, etc.) and URL pattern ("Route"), and methods to specify what template ("view") engine is used, where template files are located, and what template to use to render a response.
* You can use Express middleware to add support for cookies, sessions, and users, getting POST/GET parameters, etc. You can use any database mechanism supported by Node (Express does not define any database-related behavior).

**Body-parser**

* Body-parser is the Node.js body parsing middleware. It is responsible for parsing the incoming request bodies in a middleware before you handle it.

**Steps to create GUI application in node JS**

**Step 1: Create Node App**

* **run below command and create node app on node js command prompt**

mkdir demo

cd demo

npm init

**Step 2: Install required modules**

**• run below command and create node app on node js command prompt**

npm install express

npm install body-parser

**Step 3: Create app.js file**

const express = require('express');

var bodyParser = require('body-parser')

const app = express();

var urlencodedParser = bodyParser.urlencoded({ extended: false })

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

res.sendFile(\_\_dirname + '/index.html');

});

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

console.log('Got body:', req.body);

var first\_name= req.body.first\_name;

var last\_name= req.body.last\_name;

var email= req.body.email;

res.send("first name: "+first\_name +" last\_name: "+last\_name +" email: "+email);

});

app.listen(3000);

**Step 4: Create index.html file**

<!DOCTYPE html>

<html lang="en">

<head>

<title>Node JS Express Form Submission Example </title>

</head>

<body>

<div class="container">

<h1>Node JS Express Form Submission Example</h1>

<form action="/" method="POST">

<div>

<label>First Name</label>

<input type="text" name="first\_name">

</div>

<div>

<label>Last Name</label>

<input type="text" name="last\_name">

</div>

<div>

<label>Email address</label>

<input type="email" name="email">

</div>

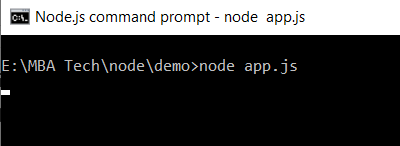
<button type="submit" name="submit">Submit</button>

</form>

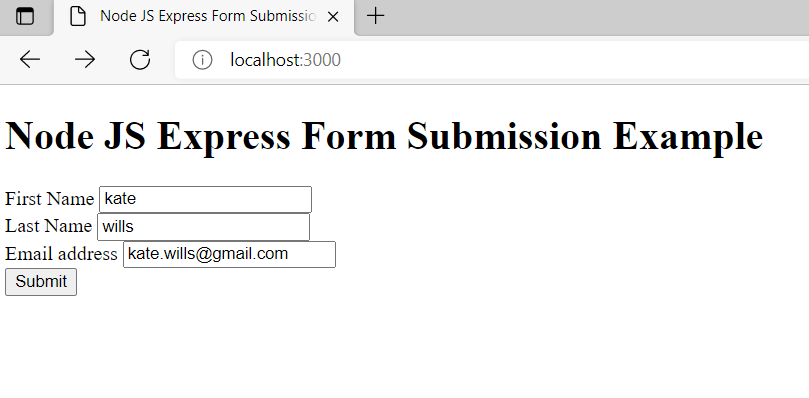
</body>

</html>

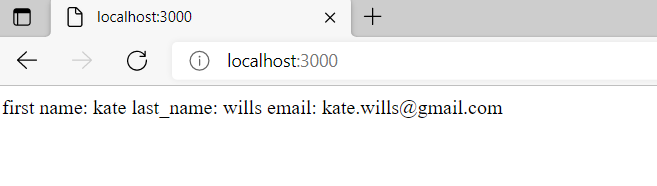
**Step 5: Run the application on node js command prompt**

****

**Step 6: open the application in browser**

****

**When you click on submit button following output will be displayed**

****

**Program**

Implement the concept of Node JS.

1. **Built-in modules and create new modules**
2. Create a module which performs all the arithmetic operations on a given numbers (addition, subtraction, multiplication and division and modulus). Include the module using require and check its operations.
3. Create an application which uses http built-in module and prints the message as an output
4. Create an application which uses fs built-in module and read the content of text file and display the same.
5. **To create GUI based web application using Express module**
6. Create a GUI based application which accepts student roll no, name, marks for 3 subjects and calculates the percentage and display all the details.

**Procedure:**

1. Open notepad editor or visual studio editor and write the code.
2. Write Source Code
3. Run in browser

Note: Doc/pdf file should be saved with Rollno\_practical8

**Part B (to be completed by students)**

**(Students must submit the soft copy as per the following segments.)**

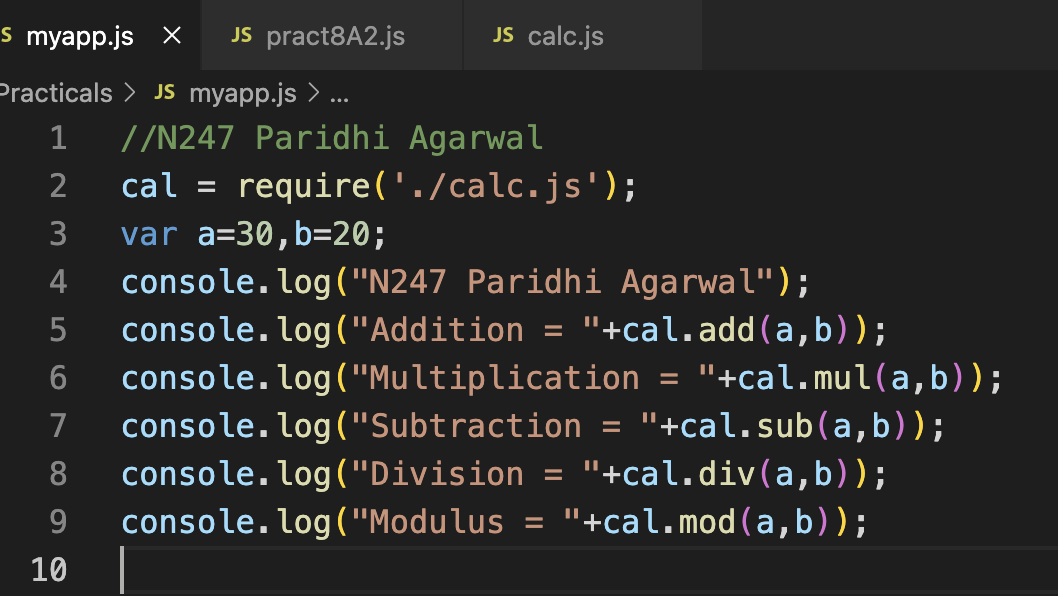
| **Roll no.: N209** | **Name: NITIGYA LADDHA** |
| --- | --- |
| **Class: MBATech CE** | **Batch: B1** |
| **Date of Experiment:19/09/2022** | **Date of Submission:19/09/2022** |

1. **Program scenario and Program code: (**Paste you program code )

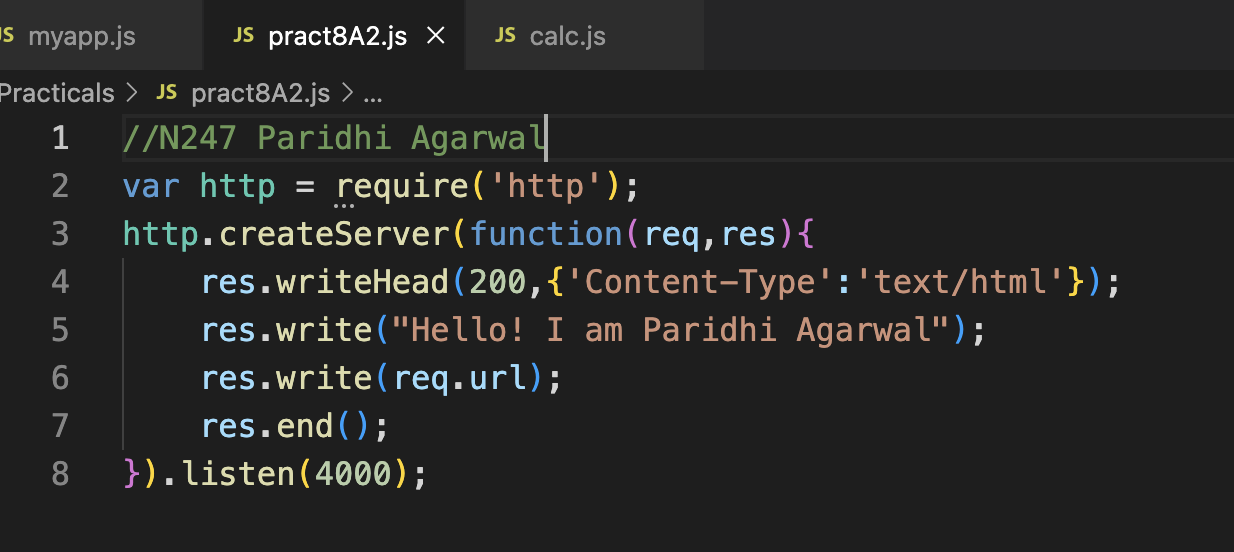
**Part A**

**1)**

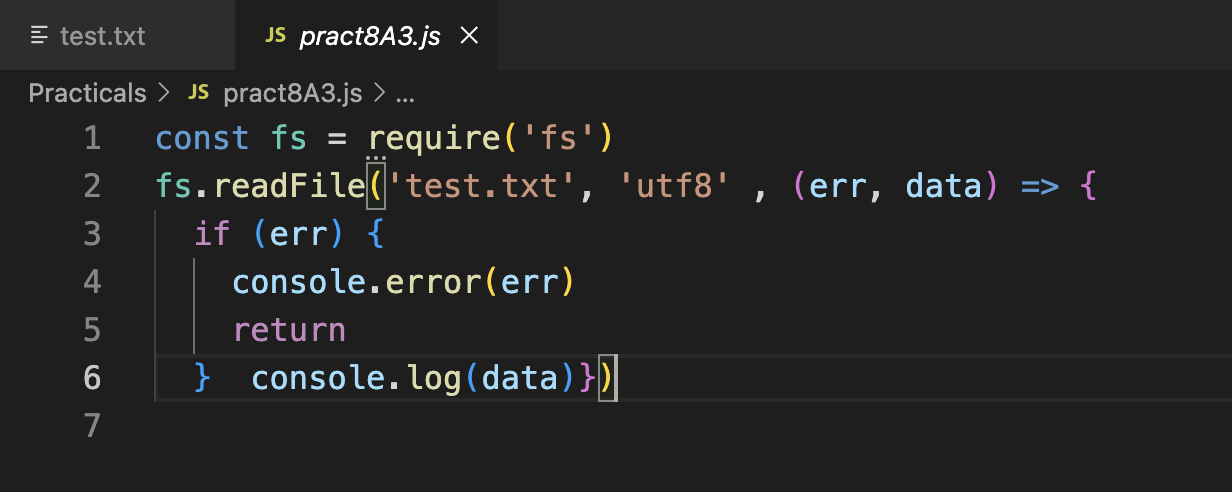
****

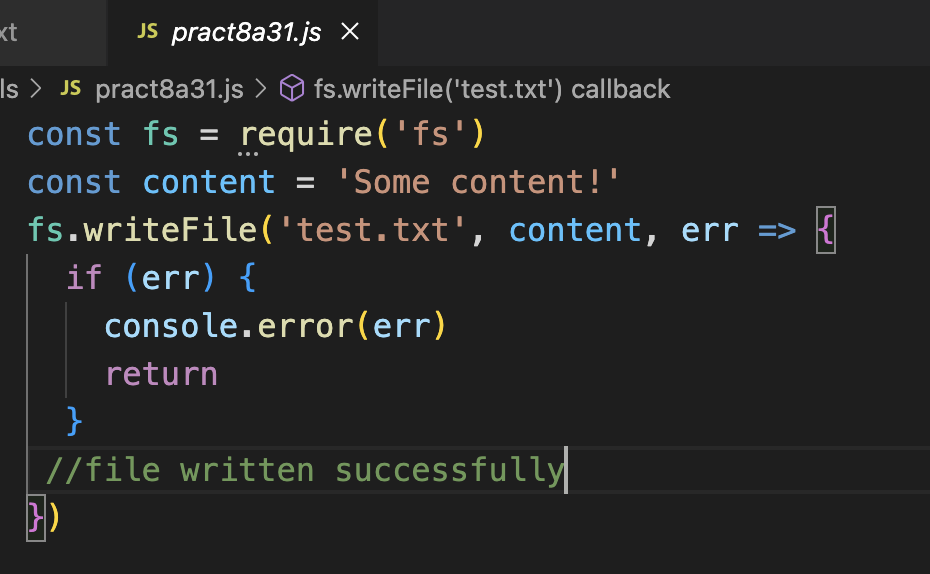
****

**2)**

****

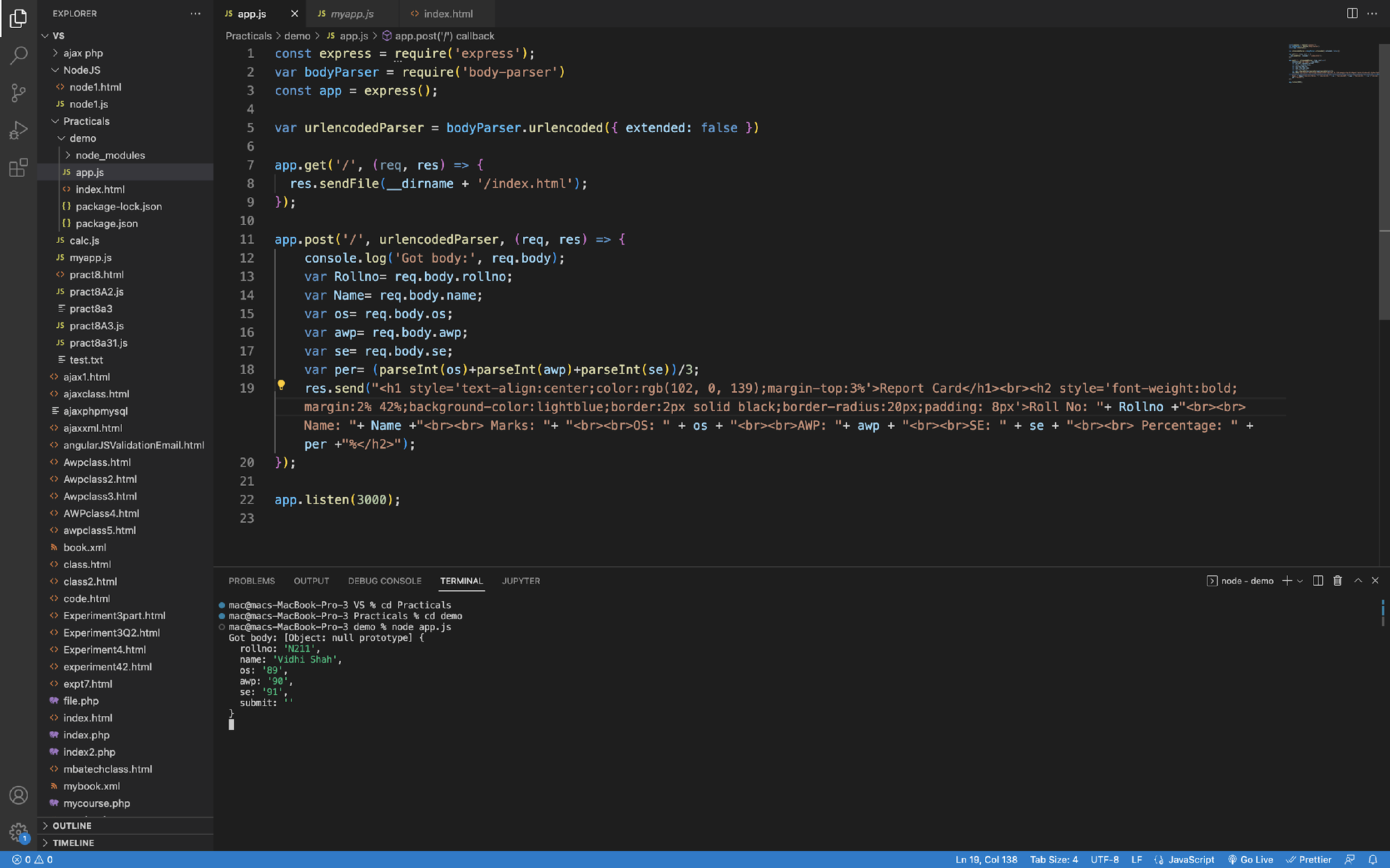
**3)**

****

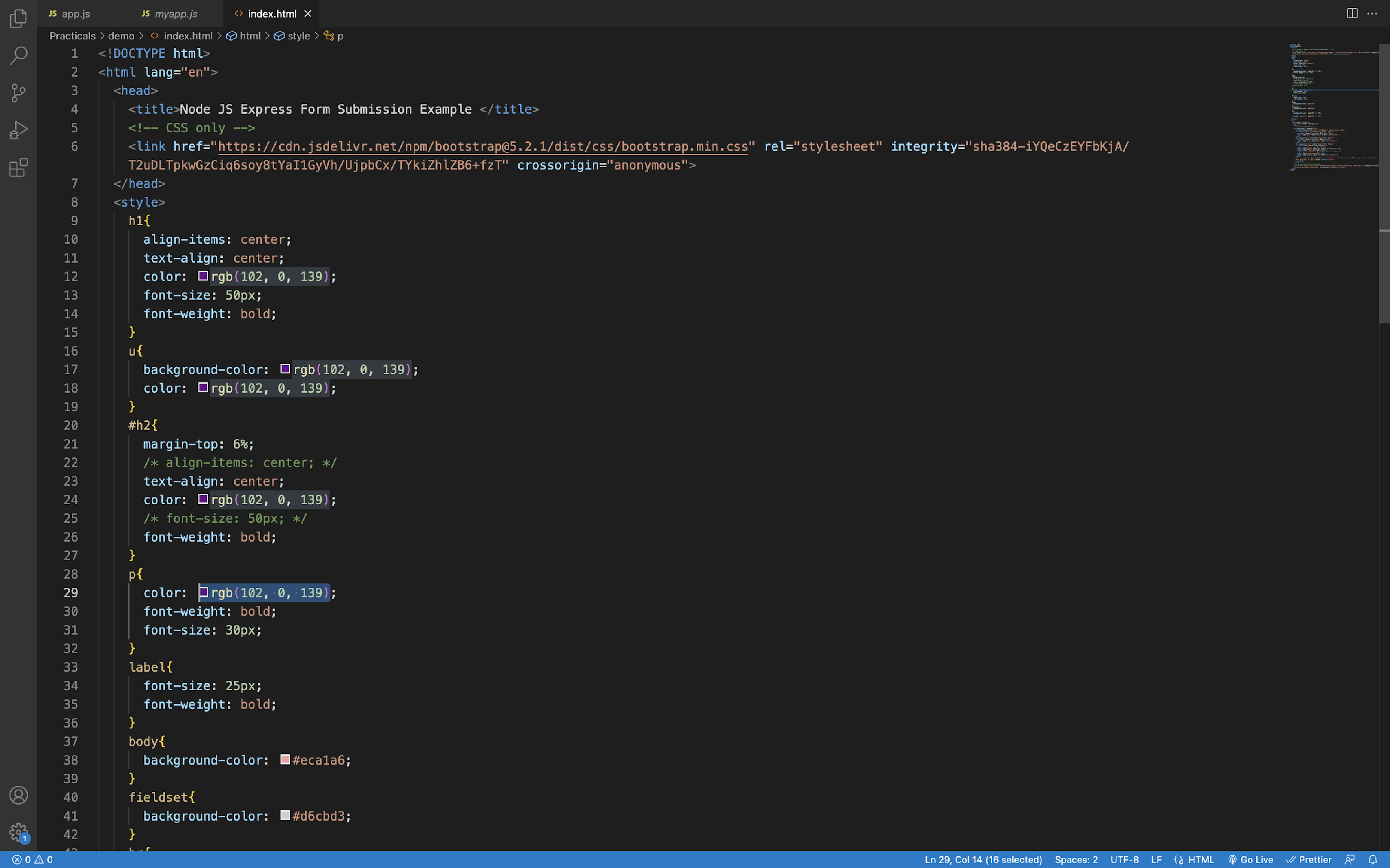
****

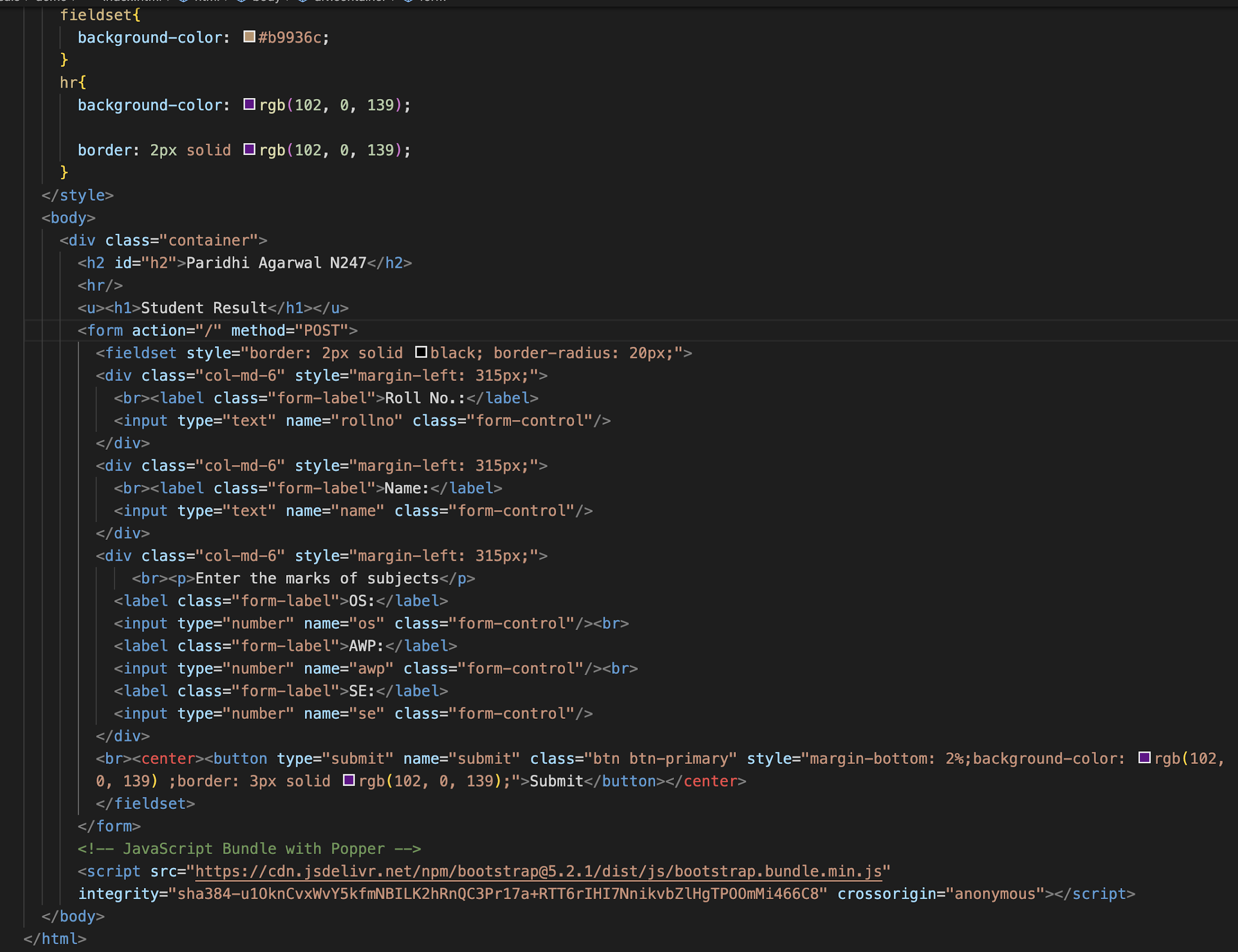
**Part B**

**App.js**

****

**Index.js**

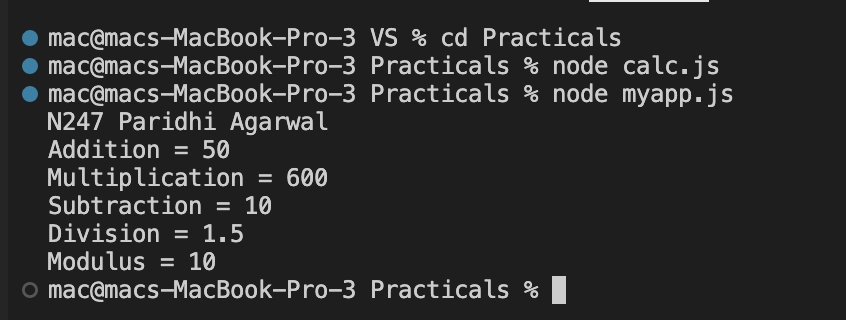
****

****

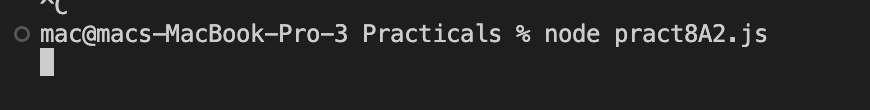
1. **Output: (**Paste your program input and output for the program)

**Part A**

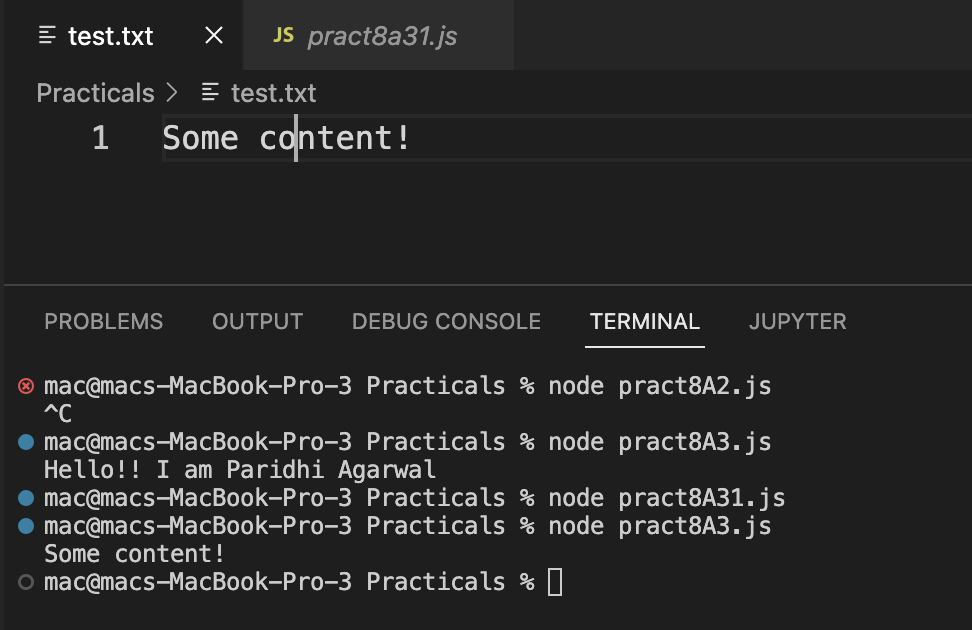
**1)**

****

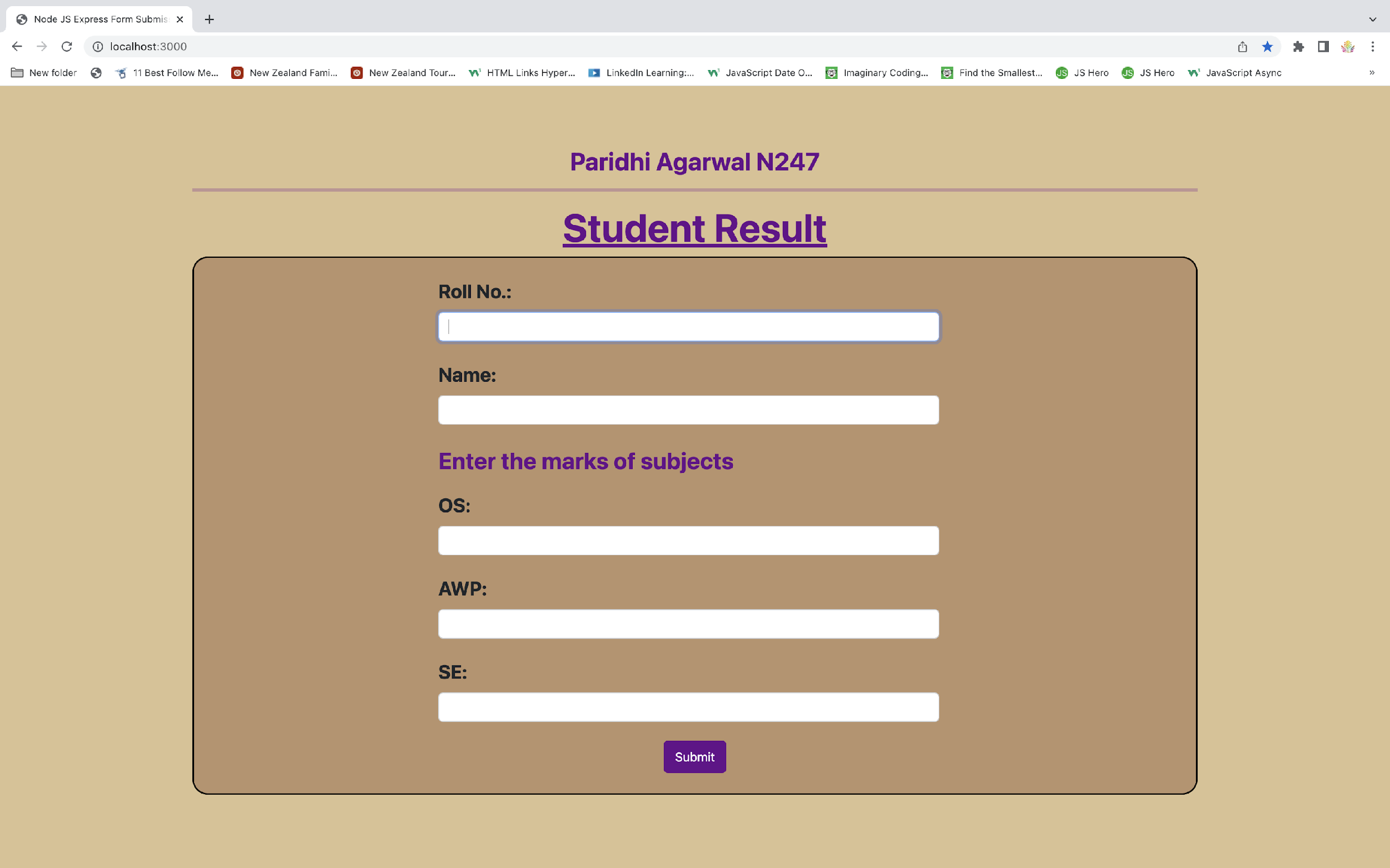
**2)**

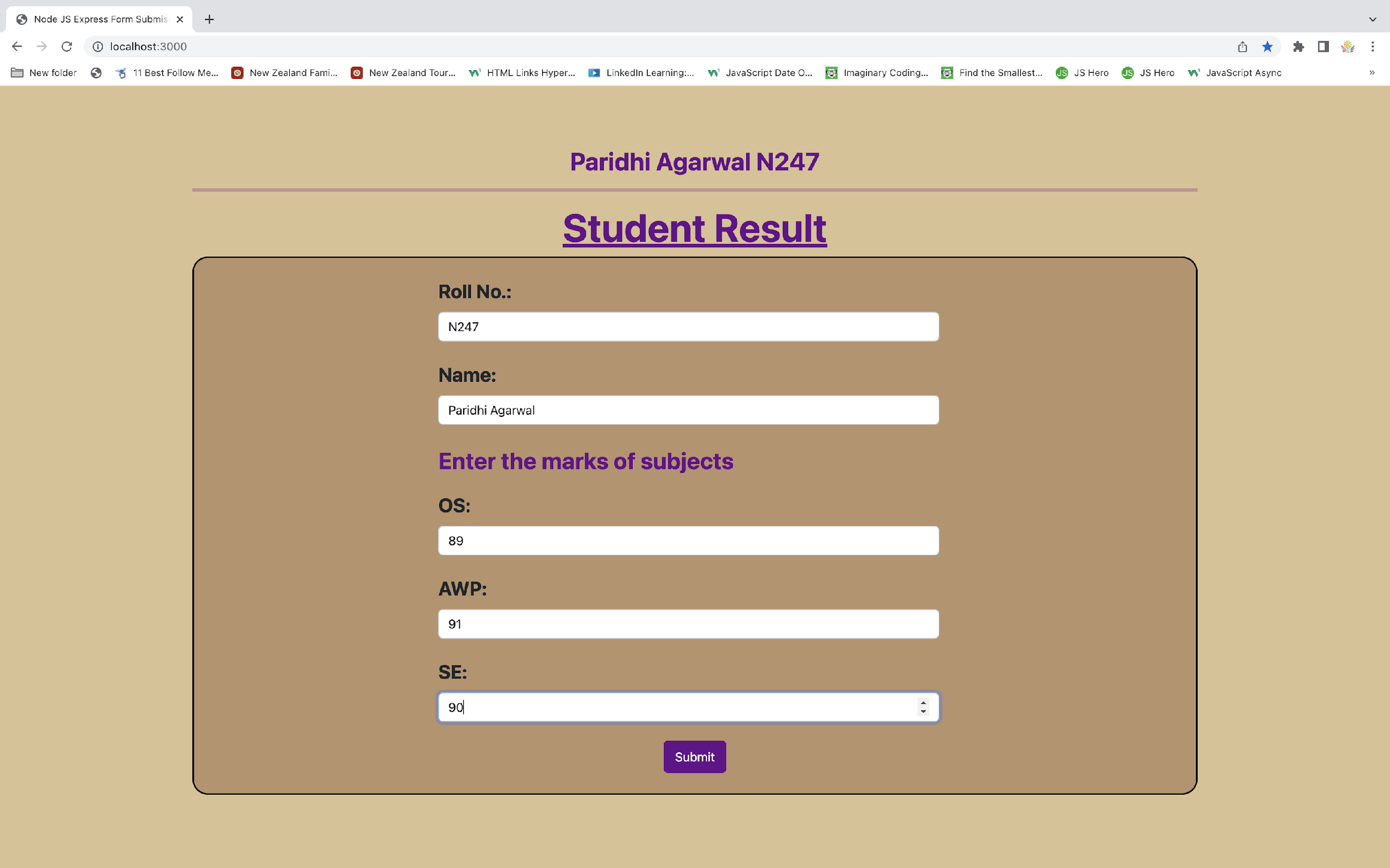
****

**3)**

****

**Part B**

****

****

1. **Observation learning and conclusion:** mention what learning you got out of practical

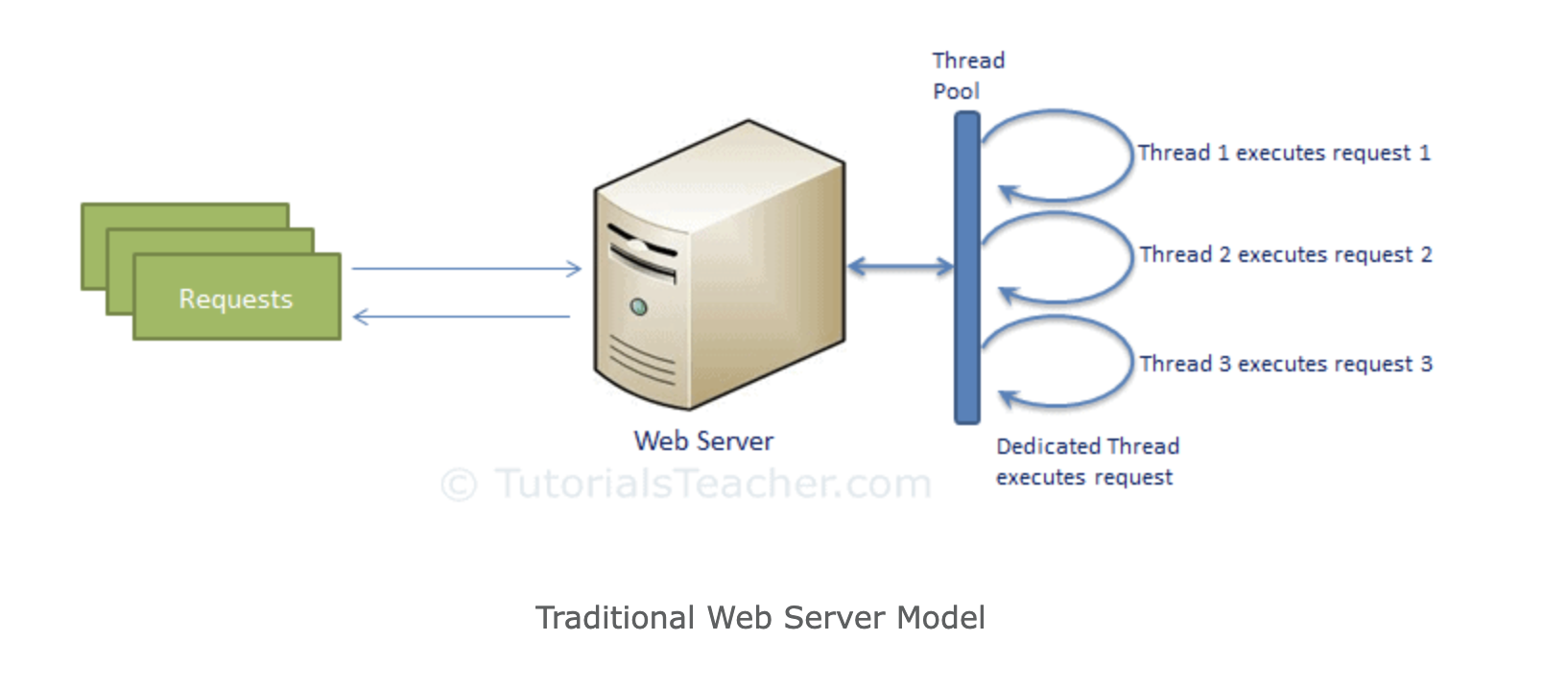
Learned and observed about node js and express js and successfully implemented it.

1. **Questions:**
   1. **Explain working of Node JS and compare how it is different from traditional web application.**

Node based server uses a single threaded model and can services much larger number of requests than traditional server like Apache HTTP Server.

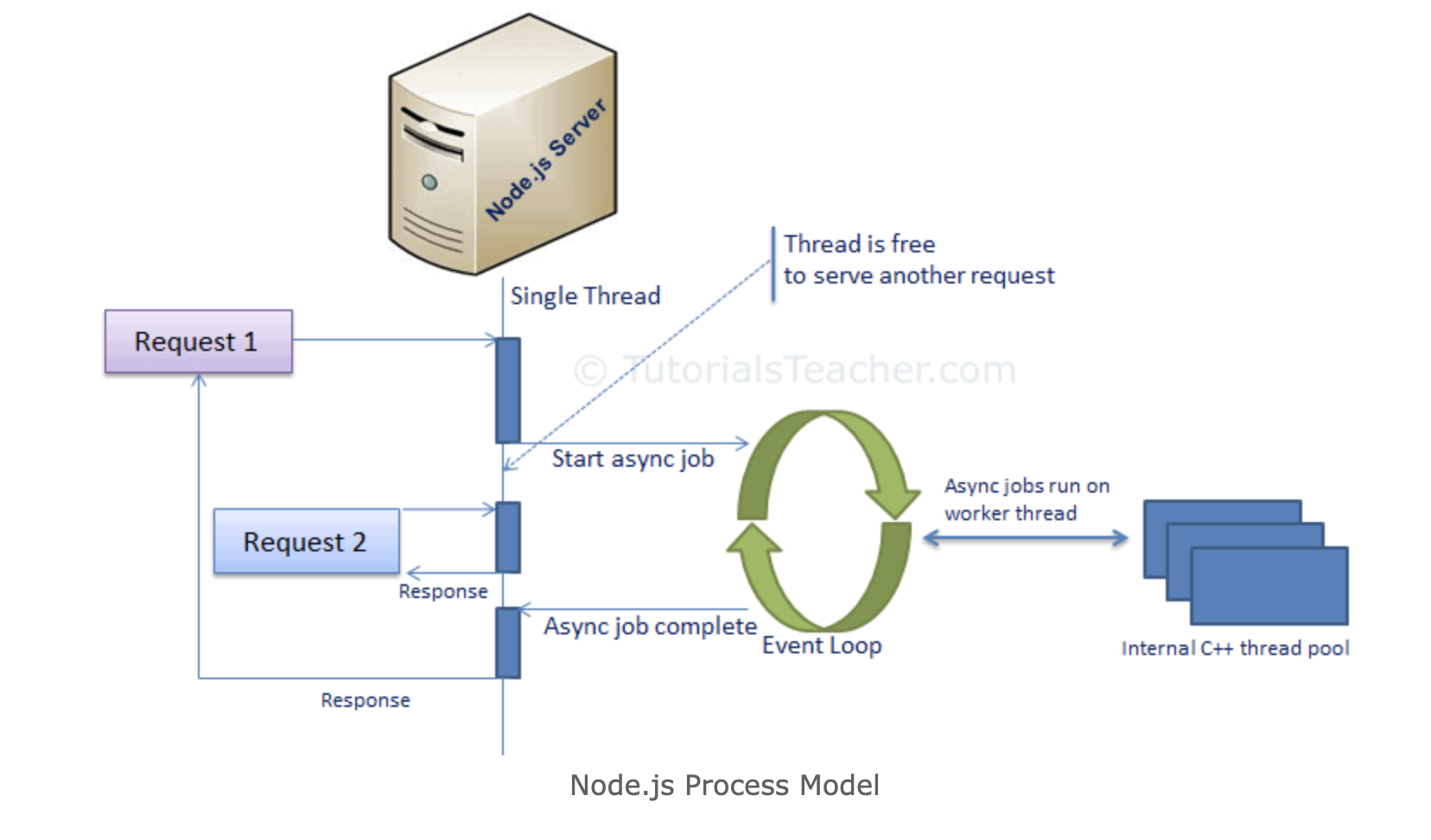
Traditional Web Server Model:

In the traditional web server model, each request is handled by a dedicated thread from the thread pool. If no thread is available in the thread pool at any point of time then the request waits till the next available thread. Dedicated thread executes a particular request and does not return to thread pool until it completes the execution and returns a response.



Node.js Process Model:

Node.js processes user requests differently when compared to a traditional web server model. Node.js runs in a single process and the application code runs in a single thread and thereby needs less resources than other platforms. All the user requests to your web application will be handled by a single thread and all the I/O work or long running job is performed asynchronously for a particular request. So, this single thread doesn't have to wait for the request to complete and is free to handle the next request. When asynchronous I/O work completes then it processes the request further and sends the response. An event loop is constantly watching for the events to be raised for an asynchronous job and executing callback function when the job completes. Internally, Node.js uses [libev](http://software.schmorp.de/pkg/libev.html) for the event loop which in turn uses internal C++ thread pool to provide asynchronous I/O.



* 1. **Explain any 4 methods of FS module**

Reading File

Use fs.readFile() method to read the physical file asynchronously.

Syntax:

fs.readFile(fileName [,options], callback)

Parameter Description:

* filename: Full path and name of the file as a string.
* options: The options parameter can be an object or string which can include encoding and flag. The default encoding is utf8 and default flag is "r".
* callback: A function with two parameters err and fd. This will get called when readFile operation completes.

Writing File

Use fs.writeFile() method to write data to a file. If file already exists then it overwrites the existing content otherwise it creates a new file and writes data into it.

Syntax:

fs.writeFile(filename, data[, options], callback)

Parameter Description:

* filename: Full path and name of the file as a string.
* Data: The content to be written in a file.
* options: The options parameter can be an object or string which can include encoding, mode and flag. The default encoding is utf8 and default flag is "r".
* callback: A function with two parameters err and fd. This will get called when write operation completes.

Open File

Alternatively, you can open a file for reading or writing using fs.open() method.

Syntax:

fs.open(path, flags[, mode], callback)

Parameter Description:

* path: Full path with name of the file as a string.
* Flag: The flag to perform operation
* Mode: The mode for read, write or readwrite. Defaults to 0666 readwrite.
* callback: A function with two parameters err and fd. This will get called when file open operation completes.

Delete File

Use fs.unlink() method to delete an existing file.

Syntax

fs.unlink(path, callback);

* 1. **Explain the use of express module**

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

Express provides methods to specify what function is called for a particular HTTP verb (GET, POST, SET, etc.) and URL pattern ("Route"), and methods to specify what template ("view") engine is used, where template files are located, and what template to use to render a response.

You can use Express middleware to add support for cookies, sessions, and users, getting POST/GET parameters, etc. You can use any database mechanism supported by Node (Express does not define any database-related behavior).