

6d.Course Name: Node.js

Module Name: Restarting Node Application

Write a program to show the workflow of restarting a Node application

Nodemon

Nodemon is a command-line utility that can be executed from the terminal. It provides a different way to start a Node.js application. It watches the application and whenever any change is detected, it restarts the application. It is very easy to get started with this tool. To install it in the application, run the below command.

Syntax: npm install nodemon -g / npm install nodemon -g

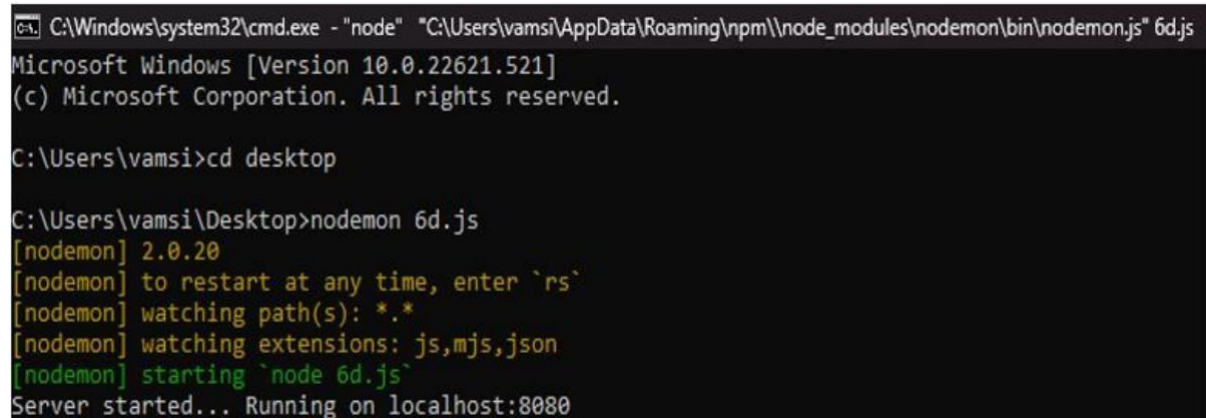
Once the 'nodemon' is installed in the machine, the Node.js server code can be executed by replacing the command "node" with "nodemon".

Syntax: nodemon app.js

program:

```
const http = require("http");  
var server = http.createServer((req, res) => {  
  res.write("I have created the server!");  
  res.end();  
});  
server.listen(8080);  
console.log("Server started... Running on localhost:8080");
```

output:



```
C:\Windows\system32\cmd.exe - "node" "C:\Users\vamsi\AppData\Roaming\npm\node_modules\nodemon\bin\nodemon.js" 6d.js  
Microsoft Windows [Version 10.0.22621.521]  
(c) Microsoft Corporation. All rights reserved.  
  
C:\Users\vamsi>cd desktop  
  
C:\Users\vamsi\Desktop>nodemon 6d.js  
[nodemon] 2.0.20  
[nodemon] to restart at any time, enter `rs`  
[nodemon] watching path(s): *.*  
[nodemon] watching extensions: js,mjs,json  
[nodemon] starting `node 6d.js`  
Server started... Running on localhost:8080
```

**Modified program:**

```
const http = require("http");  
var server = http.createServer((req, res) => {  
  res.write("I have modified the server!");  
  res.end();  
});  
server.listen(8080);  
console.log("Server started... Running on localhost:8080");
```

output:

```
C:\Users\vamsi>Select C:\Windows\system32\cmd.exe - "node" "C:\Users\vamsi\AppData\Roaming\npm\node_modules\nodemon\bin\nodemon.js" 6d.js  
Microsoft Windows [Version 10.0.22621.521]  
(c) Microsoft Corporation. All rights reserved.  
  
C:\Users\vamsi>cd desktop  
  
C:\Users\vamsi\Desktop>nodemon 6d.js  
[nodemon] 2.0.20  
[nodemon] to restart at any time, enter `rs`  
[nodemon] watching path(s): *.*  
[nodemon] watching extensions: js,mjs,json  
[nodemon] starting `node 6d.js`  
Server started... Running on localhost:8080  
[nodemon] restarting due to changes...  
[nodemon] starting `node 6d.js`  
Server started... Running on localhost:8080
```



Exp No:

Date:

Page No:

6e.Course Name: Node.js

Module Name: File Operations

Create a text file src.txt and add the following data to it. Mongo, Express, Angular, Node

Create Files:

The File System module has methods for creating new files:

1.fs.appendFile()

2.fs.open()

3. fs.writeFile()

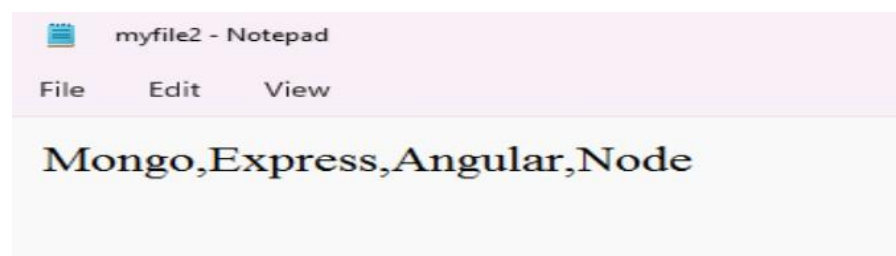
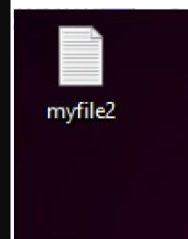
- The fs.appendFile() method appends specified content to a file. If the file does not exist, the file will be created.
- The fs.writeFile() method replaces the specified file and content if it exists. If the file does not exist, a new file, containing the specified content, will be created.
- The fs.open() method takes a "flag" as the second argument, if the flag is "w" for "writing", the specified file is opened for writing. If the file does not exist, an empty file is created.

Program:

```
var fs = require('fs');  
fs.appendFile('myfile2.txt', 'Mongo,Express,Angular,Node', function (err) {  
  if (err) throw err;  
  console.log('Saved!');  
});
```

Output:

```
C:\Windows\system32\cmd.exe  
Microsoft Windows [Version 10.0.22621.521]  
(c) Microsoft Corporation. All rights reserved.  
  
C:\Users\vamsi>cd desktop  
  
C:\Users\vamsi\Desktop>node 6e.js  
Saved!  
  
C:\Users\vamsi\Desktop>_
```



**7.a Course Name: Express.js**

Module Name: Defining a route, Handling Routes, Route Parameters, Query Parameters Implement routing for the AdventureTrails application by embedding the necessary code in the routes/route.js file.

Description:

Routing: 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.

Syntax: `router.method(path,handler)`

router: express instance or router instance

method: one of the HTTP verbs

path: is the route where request runs

handler: is the callback function that gets triggered whenever a request comes to a particular path for a matching request type .

Program:**//myNotes.js File**

```
exports.packages = async (req, res) => {  
  try {  
    res.status(200).json({  
      message: 'You can now get the requested notes for your request',  
    });  
  } catch (err) {  
    res.status(404).json({  
      status: 'fail',  
      message: err,  
    });  
  }  
};
```

```
exports.bookpackage = async (req, res) => {  
  try {  
    res.status(201).json({  
      data: 'New booking added for the POST request',  
    });  
  } catch (err) {  
    res.status(404).json({  
      status: 'fail',  
      message: err.errmsg,  
    });  
  }  
};
```



```
exports.invalid = async (req, res) => {  
  res.status(404).json({  
    status: 'fail',  
    message: 'Invalid path',  
  });  
};
```

Routing/route.js File

```
const express = require('express');  
  
const routing = express.Router();  
const notesController = require('../Controller/myNotes');  
  
routing.get('/packages', notesController.packages);  
  
routing.post('/bookpackage', notesController.bookpackage);  
  
routing.all('*', notesController.invalid);  
  
module.exports = routing;
```

//App.js File

```
const express = require('express');  
const route = require('./routes/route');  
  
const app = express();  
app.use('/', route);  
  
const port = process.env.PORT || 3000;  
app.listen(port, () => {  
  console.log(`App running on port ${port}...`);  
});
```



Exp No:

Date:

Page No:

Output:


```
C:\WINDOWS\system32\cmd. x + v

C:\Users\LENOVO\Desktop\New folder\expressjs>node app.js
App running on port 3000...
|
```

```
127.0.0.1:3000/packages x +

127.0.0.1:3000/packages

{"message":"You can now get the requested notes for your request "}
```



```
127.0.0.1:3000 x +

127.0.0.1:3000

{"status":"fail","message":"Invalid path"}
```

**7.b Course Name: Express.js**

Module Name: How Middleware works, Chaining of Middlewares, Types of Middlewares In myNotes application: (i) we want to handle POST submissions. (ii) display customized error messages. (iii) perform logging.

Description:

A middleware can be defined as a function for implementing different cross-cutting concerns such as authentication, logging, etc.

The main arguments of a middleware function are the **request** object, **response** object, and also the **next** middleware function defined in the application.

A function defined as a middleware can execute any task mentioned below:

- Any code execution.
- Modification of objects - request and response.
- Call the next middleware function.
- End the cycle of request and response.

Example to Define a MiddleWare.

```
const mylogger = async (req, res, next) => {  
  console.log(new Date(), req.method, req.url);  
  next();  
};
```

Program:**//Route1.js file**

```
const express = require('express');  
const router = express.Router();  
const myController = require('./Controller/myNotes1');  
router.get('/', myController.myMethod);  
router.get('/about', myController.aboutMethod);  
module.exports = router;
```

//myNotes.js File

```
exports.myMethod = async (req, res, next) => {  
  res.send('<h1>Welcome</h1>');  
};  
exports.aboutMethod = async (req, res, next) => {  
  res.send('<h1>About Us Page</h1>');  
};
```

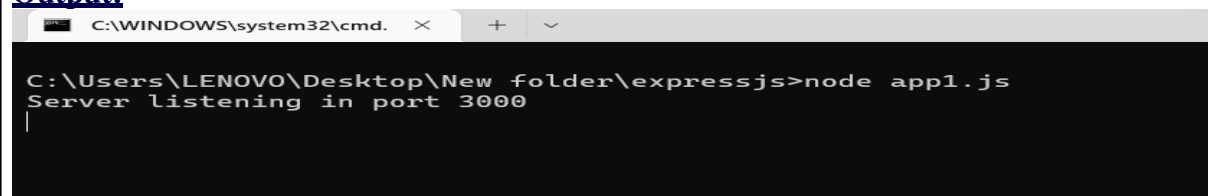
//app1.js File

```
const express = require('express');  
const router = require('./Routes/route1');  
const app = express();
```



```
const mylogger = function (req, res, next) {  
  console.log(`Req method is ${req.method}`);  
  console.log(`Req url is ${req.url}`);  
  next();  
};  
app.use(mylogger);  
app.use('/', router);  
app.listen(3000);  
console.log('Server listening in port 3000');
```

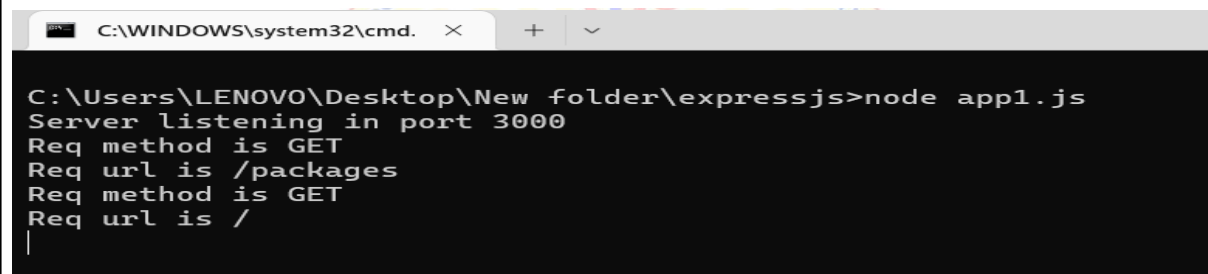
Output:



```
C:\WINDOWS\system32\cmd.  X + v  
  
C:\Users\LENOVO\Desktop\New folder\expressjs>node app1.js  
Server listening in port 3000  
|
```



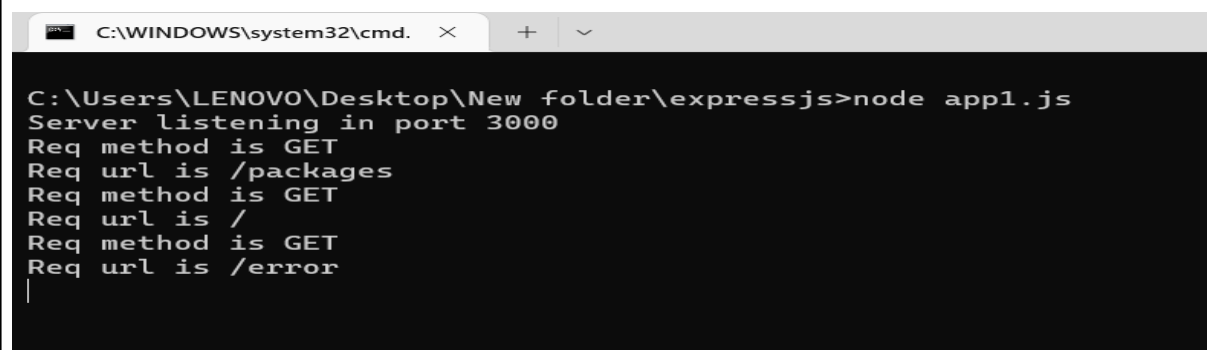
We are loading Default



```
C:\WINDOWS\system32\cmd.  X + v  
  
C:\Users\LENOVO\Desktop\New folder\expressjs>node app1.js  
Server listening in port 3000  
Req method is GET  
Req url is /packages  
Req method is GET  
Req url is /  
|
```



```
{"status": "fail", "message": "WE ARE DISPLAYING OUR OWN ERROR MESSAGE"}
```



```
C:\WINDOWS\system32\cmd.  X + v  
  
C:\Users\LENOVO\Desktop\New folder\expressjs>node app1.js  
Server listening in port 3000  
Req method is GET  
Req url is /packages  
Req method is GET  
Req url is /  
Req method is GET  
Req url is /error  
|
```


7.c Course Name: Express.js

Module Name: Connecting to MongoDB with Mongoose, Validation Types and Defaults

Write a Mongoose schema to connect with MongoDB

Description:

Before we get into the specifics of validation syntax, please keep the following rules in mind:

- Validation is defined in the SchemaType
- Validation is middleware. Mongoose registers validation as a pre('save') hook on every schema by default.
- You can disable automatic validation before save by setting the validateBeforeSave option
- You can manually run validation using doc.validate(callback) or doc.validateSync()
- You can manually mark a field as invalid (causing validation to fail) by using doc.invalidate(...)
- Validators are not run on undefined values. The only exception is the required validator.

Program:

```
const express=require('express')
const mongoose=require('mongoose')
const app=express()
app.listen(3000,()=>console.log(" server running. ...."))
const
url="mongodb+srv://mstdatabase:mstdatabase@cluster0.xx7bb4u.mongodb.net/?retryWrites=true&w
=majority";

mongoose.connect(url).then(()=>console.log("Database
Connected. ...")).catch(err=>console.log(err));
```

Output:

```
D:\mstd>node app.js
server running.....
Database Connected.....
█
```





Exp No:
Date:

Page No:

Creating schema

```
const express=require('express')
const mongoose=require('mongoose')
const app=express()
app.listen(3000,()=>console.log(" Server running..... "))
const
url="mongodb+srv://mstdatabase:mstdatabase@cluster0.xx7bb4u.mongodb.net/?retryWrites=true&w=
=majority";
mongoose.connect(url).then(=>console.log("Database
Connected. ...")).catch(err=>console.log(err));
var bookSchema = mongoose.Schema({
    name: String,
    isbn: {type: String, index: true},
    author: String,
    pages: Number
});
```

Output:

```
D:\mstd>node app.js
Server running.....
Database Connected.....
```

7.d Course Name: Express.js

Module Name: Models

Write a program to wrap the Schema into a Model object

Description: Schema wrapping (@graphql-tools/wrap) creates a modified version of a schema that proxies, or "wraps", the original unmodified schema. This technique is particularly useful when the original schema cannot be changed, such as with remote schemas.

Schema wrapping works by creating a new "gateway" schema that simply delegates all operations to the original subschema. A series of transforms are applied that may modify the shape of the gateway schema and all proxied operations; these operational transforms may modify an operation prior to delegation, or modify the subschema result prior to its return.

Program:

```
const express=require('express')
const mongoose=require('mongoose')
const url="mongodb://0.0.0.0:27017/Hell";
mongoose.connect(url,{useNewUrlParser:true},{useUnifiedTopology:true}).then(=>console.log("Database Connected....")).catch(err=>console.log(err));
var bookSchema = mongoose.Schema({
  name: String,
  isbn: {type: String, index: true},
  author: String,
  pages: Number
});
var Book = mongoose.model("Book", bookSchema);
var db = mongoose.connection;
db.on("error", console.error.bind(console, "connection error:"));
db.once("open", function(){
  console.log("Connected to DB");
});
```

Output:

```
D:\mstd>node app.js
Connected to DB
Database Connected.....
|
```

8.a Course Name: Express.js

Module Name: CRUD Operations

Write a program to perform various CRUD (Create-Read-Update-Delete) operations using Mongoose library functions

Description: CRUD OPERATIONS

Create: We'll be setting up a post request to '/save' and we'll create a new student object with our model and pass with it the request data from Postman. Once this is done, we will use .save() to save it to the database.

Retrieve: To retrieve records from a database collection we make use of the .find() function.

Update: Just like with the delete request, we'll be using the _id to target the correct item. .findByIdAndUpdate() takes the target's id, and the request data you want to replace it with.

Program:

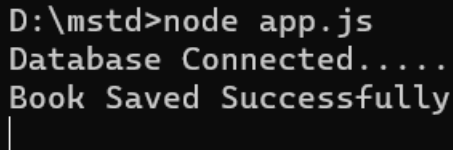
Create:

```
const express=require('express')
const mongoose=require('mongoose')
const url="mongodb://0.0.0.0:27017/Hell";
mongoose.connect(url,{useNewUrlParser:true},{useUnifiedTopology:true}).then(()=>console.log("Database Connected... ")).catch(err=>console.log(err));
var bookSchema = mongoose.Schema({
  name: String,
  isbn: {type: String, index: true},
  author: String,
  pages: Number
});
var Book = mongoose.model("Book", bookSchema);

var db = mongoose.connection;
var book1 = new Book({
  name:"Mongoose Demo 1",
  isbn: "MNG123",
  author: "Author1, Author2",
  pages: 123
});
```



```
book1.save(function(err){  
    if ( err )  
        throw err;  
    console.log("Book Saved Successfully");  
});
```



```
D:\mstd>node app.js  
Database Connected.....  
Book Saved Successfully  
|
```

Read

```
const express=require('express')  
const mongoose=require('mongoose')  
const url="mongodb://0.0.0.0:27017/Hell";  
mongoose.connect(url,{useNewUrlParser:true},{useUnifiedTopology:true}).then(()=>console.log("Database Connected.... ")).catch(err=>console.log(err));  
var bookSchema = mongoose.Schema({  
    name: String,  
    isbn: {type: String, index: true},  
    author: String,  
    pages: Number  
});  
var Book = mongoose.model("Book", bookSchema);  
  
var db = mongoose.connection;  
var queryBooks = function(){  
  
    Book.find( function(err, result){
```



```
if ( err )  
  
    throw err;  
  
console.log("Find Operations: " + result);  
  
});  
  
}  
  
queryBooks();
```

Output:

```
Database Connected....  
Find Operations: {  
  _id: new ObjectId("636cf1bb7a291be85ed8067d"),  
  name: 'Mongoose Demo 1',  
  isbn: 'MNG123',  
  author: 'Author1, Author2',  
  pages: 123,  
  __v: 0  
}, {  
  _id: new ObjectId("636cf36e1e0e614dbb5a0089"),  
  name: 'Mongoose Demo 1',  
  isbn: 'MNG123',  
  author: 'Author1, Author2',  
  pages: 123,  
  __v: 0  
}, {  
  _id: new ObjectId("636cf473e450b49c8bc26843"),  
  name: 'Mongoose Demo 1',  
  isbn: 'MNG123',  
  author: 'Author1, Author2',  
  pages: 123,  
  __v: 0  
}, {  
  _id: new ObjectId("636cf473e450b49c8bc26844"),  
  name: 'JAvA',  
  isbn: 'Hello',  
  author: 'kello',  
  pages: 123,  
  __v: 0  
}, {  
  _id: new ObjectId("636cf4a59b74e217409d7b0b"),  
  name: 'JAvA',  
  isbn: 'Hello',  
  author: 'kello',  
  pages: 123,  
  __v: 0  
}  
}
```

Update

```
const express=require('express')  
  
const mongoose=require('mongoose')  
  
const url="mongodb://0.0.0.0:27017/Hell";  
  
mongoose.connect(url,{useNewUrlParser:true},{useUnifiedTopology:true}).then(()=>console.log("Database Connected.... ")).catch(err=>console.log(err));  
  
var bookSchema = mongoose.Schema({  
  
    name: String,  
  
    isbn: {type: String, index: true},  
  
    author: String,  
  
    pages: Number  
  
});
```



```
var Book = mongoose.model("Book", bookSchema);

var db = mongoose.connection;

var updateBook = function(){
    Book.updateOne({ $name: "JAvA" }, { $set: { name: "JAVA" } }, function(err, result){
        console.log("Updated successfully");
        console.log(result);
    });
}updateBook();
```

Output:

```
C:\WINDOWS\system32\cmd. x + v

D:\mstd>node app.js
Database Connected.....
Updated successfully
{
  acknowledged: true,
  modifiedCount: 1,
  upsertedId: null,
  upsertedCount: 0,
  matchedCount: 1
}
```

Delete

```
const express=require('express')

const mongoose=require('mongoose')

const url="mongodb://0.0.0.0:27017/Hell";

mongoose.connect(url,{useNewUrlParser:true},{useUnifiedTopology:true}).then(()=>console.log("Database Connected....")).catch(err=>console.log(err));

var bookSchema = mongoose.Schema({
    name: String,
    isbn: {type: String, index: true},
    author: String,
    pages: Number});

var Book = mongoose.model("Book", bookSchema);
```



```
var db = mongoose.connection;

var deleteBook = function(){

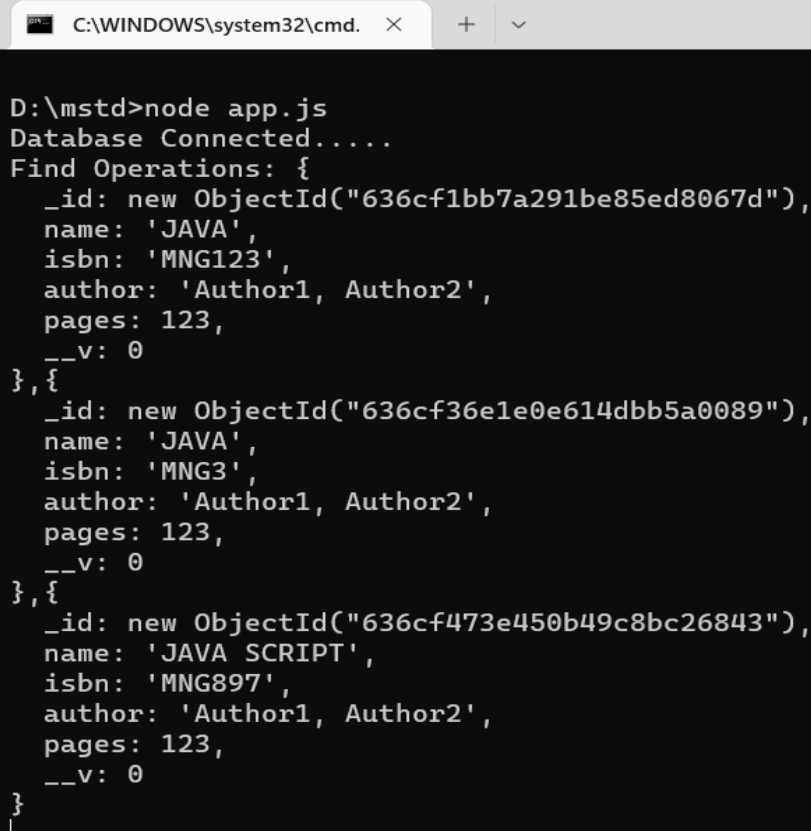
Book.deleteMany({name:"JAva"},function(err,result){if(err)

    console.log(err);

    else

    console.log("deleted"))}.exec(); }

deleteBook();
```

Afterdeletong records Database is:

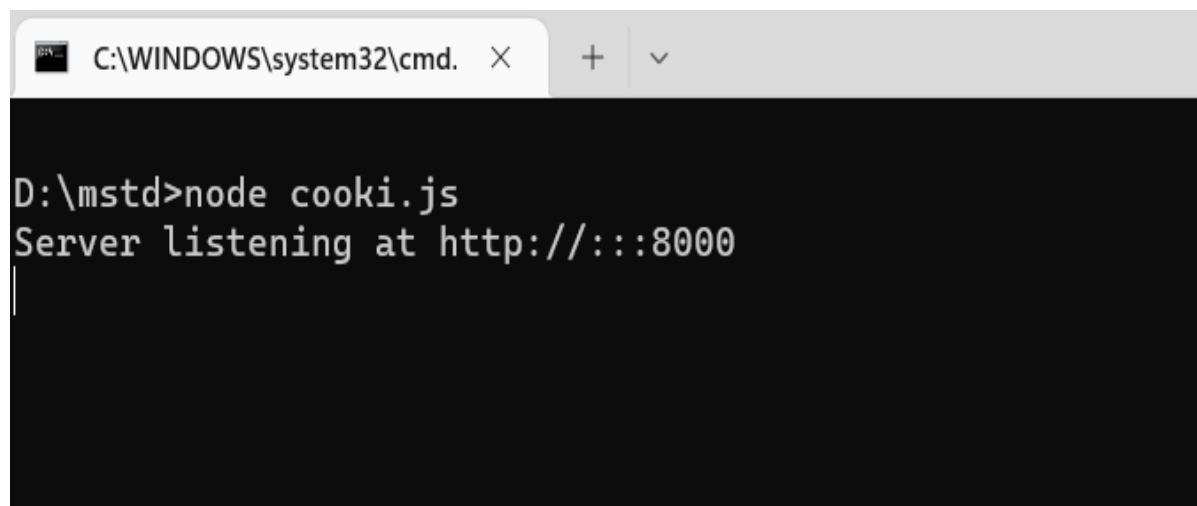
```
C:\WINDOWS\system32\cmd.  X  +  v

D:\mstd>node app.js
Database Connected.....
Find Operations: {
  _id: new ObjectId("636cf1bb7a291be85ed8067d"),
  name: 'JAVA',
  isbn: 'MNG123',
  author: 'Author1, Author2',
  pages: 123,
  __v: 0
}, {
  _id: new ObjectId("636cf36e1e0e614dbb5a0089"),
  name: 'JAVA',
  isbn: 'MNG3',
  author: 'Author1, Author2',
  pages: 123,
  __v: 0
}, {
  _id: new ObjectId("636cf473e450b49c8bc26843"),
  name: 'JAVA SCRIPT',
  isbn: 'MNG897',
  author: 'Author1, Author2',
  pages: 123,
  __v: 0
}
|
```


8.c Course Name: Express.js**Module Name: Why Session management, Cookies****Write a program to explain session management using cookies.****Program:**

```
var express = require('express');
var cookieParser = require('cookie-parser');
var app = express();
app.use(cookieParser());
app.get('/cookieset',function(req, res){
res.cookie('cookie_name', 'cookie_value');
res.cookie('College', 'Aditya');
res.cookie('Branch', 'Cse');

res.status(200).send('Cookie is set');
});
app.get('/cookieget', function(req, res) {
  res.status(200).send(req.cookies);
});
app.get('/', function (req, res) {
  res.status(200).send('Welcome to Aditya');
});
var server = app.listen(8000, function () {
  var host = server.address().address;
  var port = server.address().port;
  console.log('Server listening at http://%s:%s', host, port);
});
```

Output:

```
C:\WINDOWS\system32\cmd.  X  +  v
D:\mstd>node cooki.js
Server listening at http://:::8000
```



Exp No:
Date:

Page No:

127.0.0.1:8000 x +

127.0.0.1:8000

Welcome to Aditya

127.0.0.1:8000/cookieset x +

127.0.0.1:8000/cookieset

Cookie is set

127.0.0.1:8000/cookieget x +

127.0.0.1:8000/cookieget

```
{"cookie_name": "cookie_value", "College": "Aditya", "Branch": "Cse"}
```



Exp No:

Date:

Page No:

8.d Course Name: Express.js

Module Name: Sessions

Write a program to explain session management using sessions

Program:

```
const express = require("express")

const session = require('express-session')

const app = express()

var PORT = process.env.port || 3000

app.use(session({

secret: 'Your_Secret_Key',

resave: true,

saveUninitialized: true

}))

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

req.session.name = 'Sessionname:alr'

return res.send("Session Set")

})

app.get("/session", function(req, res){

var name = req.session.name

return res.send(name)

})

app.listen(PORT, function(error){

if(error) throw error

console.log("Server created Successfully on PORT :", PORT)

})
```



Exp No:

Date:

Page No:

Output:

```
C:\WINDOWS\system32\cmd. x + v
D:\mstd>node session.js
Server created Successfully on PORT : 3000
```

```
127.0.0.1:3000/session x +
127.0.0.1:3000/session
Sessionname:alr
```

```
127.0.0.1:3000/session x +
127.0.0.1:3000/session
{"cookie":{"originalMaxAge":null,"expires":null,"httpOnly":true,"path":"/"},"name":"Sessionname:alr"}
```

```
C:\WINDOWS\system32\cmd. x + v
D:\mstd>node session.js
Server created Successfully on PORT : 3000
Session Destroyed
```

```
127.0.0.1:3000 x +
127.0.0.1:3000
Session Set
```



Exp No:

Date:

Page No:

8.e Course Name: Express.js

Module Name:

Why and What Security, Helmet Middleware Implement security features in myNotes application.

Program:

App.js

```
const express = require('express');
const routing = require('./route');
const app = express();
app.use('/', routing);
app.listen(3000);
console.log('Server listening in port 3000');
```

route.js

```
const express = require('express');
const router = express.Router();
router.get('/', function (req, res) {
  res.send('<h1>Express</h1>');
});
router.get('/about', function (req, res) {
  res.send('About Us Page');
});
module.exports = router;
```

test.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <style>
    p {
      color: red;
    }
    iframe {
      width: 100%;
      height: 90%
    }
  </style>
</head>
```

```
</style>
</head>
<body>
  <p>Clickjacked</p>
  <iframe src="http://localhost:3000"></iframe>
</body>
</html>
```

Output:



Without using Helment

Implementing Helmet

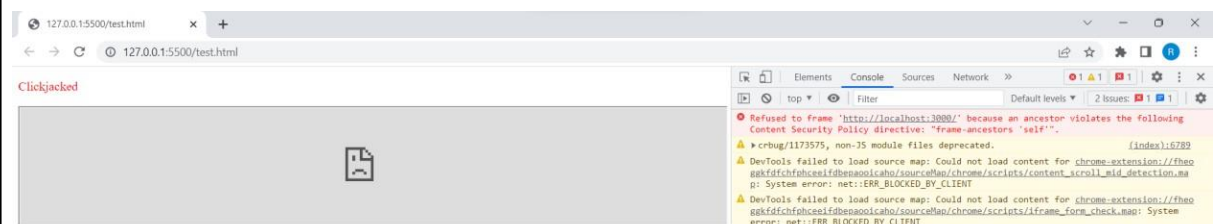
App.js

```
const express = require('express');
const helmet = require('helmet');
const routing = require('./route');
const app = express();

app.use(helmet());
app.use('/', routing);
app.listen(3000);

console.log('Server listening in port 3000');
```

Output:



**9.a Course Name: Typescript****Module Name: Basics of TypeScript**

On the page, display the price of the mobile-based in three different colors. Instead of using the number in our code, represent them by string values like GoldPlatinum, PinkGold, SilverTitanium

Program:

```
const obj:{GoldPlatinum: string}={GoldPlatinum:"$1000"}  
  
const ob1:{PinkGold: string,}={PinkGold:"$900"}  
  
const ob2:{SilverTitanium: string}={SilverTitanium:"$1500"}  
  
console.log("\nMobilecolor Price\n")  
  
console.log("\nGoldPlatinum:\t "+obj.GoldPlatinum+"\n")  
  
console.log("PinkGold:\t "+ob1.PinkGold+"\n")  
  
console.log(" SilverTitanium:\t "+ob2.SilverTitanium+"\n")
```

Output:

```
D:\mstd\typescrip>ts-node 9a.ts
```

```
Mobilecolor Price
```

```
GoldPlatinum:    $1000
```

```
PinkGold:        $900
```

```
SilverTitanium:  $1500
```



Exp No:

Date:

Page No:

9.b) Define an arrow function inside the event handler to filter the product array with the selected product object using the productId received by the function. Pass the selected product object to the next screen.

Program:

```
var getProductDetails=(productId : number):string=>{return "product Id:"+productId};  
console.log(getProductDetails(1234));
```

Output:

```
D:\mstd\typescrip>ts-node 9b.ts  
product Id: 1234  
  
D:\mstd\typescrip>
```



9.c Course Name: Typescript

Module Name: Parameter Types and Return Types

Consider that developer needs to declare a function - getMobileByVendor which accepts string as input parameter and returns the list of mobiles.

Program:

```
function getMobileByManufacturer(manufacturer: string): string[] {  
    let mobileList: string[];  
    if (manufacturer === 'Samsung') {  
        mobileList = ['Samsung Galaxy S6 Edge', 'Samsung Galaxy Note 7',  
'Samsung Galaxy J7 SM-J700F'];  
        return mobileList;  
    } else if (manufacturer === 'Apple') {  
        mobileList = ['Apple iPhone 5s', 'Apple iPhone 6s ', 'Apple iPhone 7'];  
        return mobileList;  
    } else {  
        mobileList = ['Nokia 105', 'Nokia 230 Dual Sim'];  
        return mobileList;  
    }  
}  
console.log("The available Samsung mobile list: [" + getMobileByManufacturer('Samsung')+"]");  
console.log("\nThe available Iphone mobile list: [" + getMobileByManufacturer('Apple')+"]");
```

Output:

```
D:\mstd\typescrip>ts-node 9c.ts  
The available Samsung mobile list: [Samsung Galaxy S6 Edge,Samsung Galaxy Note 7,Samsung Galaxy J7 SM-J700F]  
  
The available Iphone mobile list: [Apple iPhone 5s,Apple iPhone 6s ,Apple iPhone 7]  
D:\mstd\typescrip>
```



Exp No:

Date:

Page No:

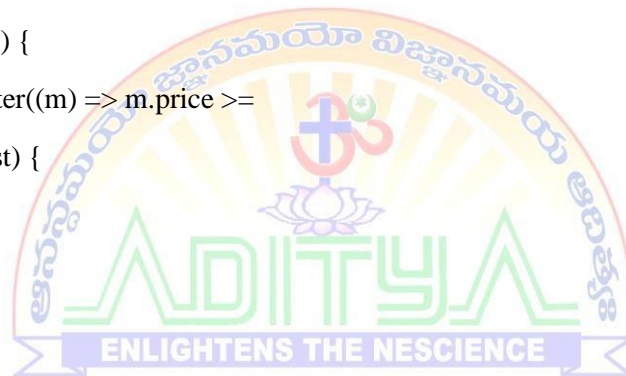
9d) Course Name: Typescript

Module Name: Arrow Function Consider that

developer needs to declare a manufacturer's array holding 4 objects with id and price as a parameter and needs to implement an arrow function - myfunction to populate the id parameter of manufacturers array whose price is greater than or equal to 150 dollars then below mentioned code snippet would fit into this requirement.

Program:

```
var manufacturers = [{ id: 'Samsung', price: 150 },
{ id: 'Microsoft', price: 200 },
{ id: 'Apple', price: 400 },
{ id: 'Micromax', price: 100 }
];
var test;
console.log('Details of Manufacturer array are :
');function myFunction() {
test = manufacturers.filter((m) => m.price >=
150);for (var item of test) {
console.log(item.id);
}
}
myFunction();
```



output:

```
D:\mstd\typescrip>ts-node 9d.ts
Details of Manufacturer array are :
Samsung
Microsoft
Apple
D:\mstd\typescrip>
```

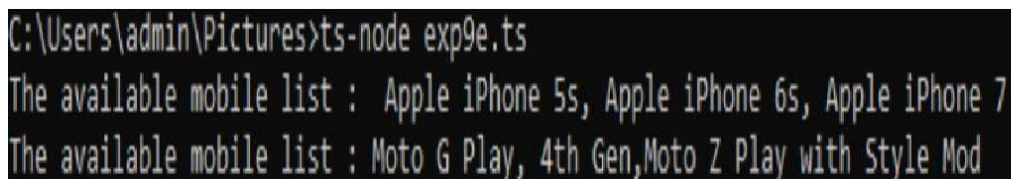
9e)Course Name: Typescript**Module Name: Optional and Default Parameters Declarea**

function - getMobileByManufacturer with two parameters namely manufacturer and id, where manufacturer value should be passed as Samsung and id parameter should be optional while invoking the function, if id is passed as 101 then this function should return Moto mobile list and if manufacturer parameter is either Samsung/Apple then this function should return respective mobile list and similar to make Samsung as default Manufacturer. Below mentioned code-snippet would fit into this requirement

Program:

```
function getMobileByManufacturer(manufacturer: string = 'Samsung', id?: number): string[] {
    let mobileList: string[];
    if (id) {
        if (id === 101) {
            mobileList = ['Moto G Play, 4th Gen', 'Moto Z Play with Style Mod'];
            return mobileList;
        }
    }
    if (manufacturer === 'Samsung') {
        mobileList = ['Samsung Galaxy S6 Edge', 'Samsung Galaxy Note 7',
            'Samsung Galaxy J7 SM-J700F'];
        return mobileList;
    } else if (manufacturer === 'Apple') {
        mobileList = ['Apple iPhone 5s', 'Apple iPhone 6s', 'Apple iPhone 7'];
        return mobileList;
    } else {
        mobileList = ['Nokia 105', 'Nokia 230 Dual Sim'];
        return mobileList;
    }
}

console.log('The available mobile list : ' + getMobileByManufacturer('Apple'));
console.log('The available mobile list : ' + getMobileByManufacturer(undefined, 101))
```

Output:


```
C:\Users\admin\Pictures>ts-node exp9e.ts
The available mobile list : Apple iPhone 5s, Apple iPhone 6s, Apple iPhone 7
The available mobile list : Moto G Play, 4th Gen, Moto Z Play with Style Mod
```



Exp No:
Date:

Page No:

10.a Course Name: Typescript

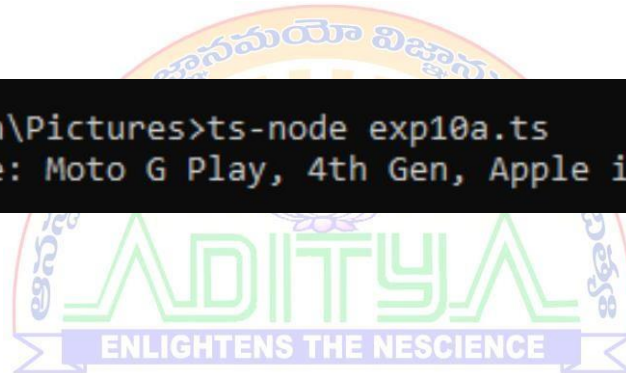
Module Name: Rest Parameter Implement business logic for adding multiple Product values into a cart variable which is type of string array.

Program:

```
const cart: string[] = [];  
const pushtoCart = (item: string) => { cart.push(item); };  
function addtoCart(...productName: string[]): string[] {  
  for (const item of productName) {  
    pushtoCart(item);  
  }  
  return cart;  
}  
console.log('Cart Items are:' + addtoCart(' Moto G Play, 4th Gen', ' Apple iPhone 5s'));
```

Output:

```
C:\Users\admin\Pictures>ts-node exp10a.ts  
Cart Items are: Moto G Play, 4th Gen, Apple iPhone 5s
```



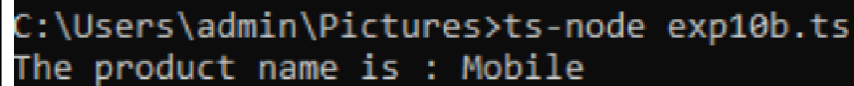
10.b Course Name: Typescript

Module Name: Creating an Interface Declare an interface named - Product with two properties like productId and productName with a number and string datatype and need to implement logic to populate the Product details.

Program:

```
interface Product {  
  productId: number ;  
  productName: string ;  
}  
  
function getProductDetails(productobj: Product): string {  
  return 'The product name is : ' + productobj.productName;  
}  
  
const prodObject = {productId: 1001, productName: 'Mobile'};  
const productDetails: string = getProductDetails(prodObject);  
console.log(productDetails);
```

Output:



```
C:\Users\admin\Pictures>ts-node exp10b.ts  
The product name is : Mobile
```

ENLIGHTENS THE NESCIENCE



Exp No:
Date:

Page No:

10.c Course Name: Typescript

Module Name: Duck Typing Declare an interface named

- Product with two properties like productId and productName with the number and string datatype and need to implement logic to populate the Product details.

Program:

```
interface Product {  
  productId: number;  
  productName:string;  
}  
  
function getProductDetails(productobj: Product): string {  
  return 'The product name is : ' + productobj.productName;  
}  
  
const prodObject = {productId: 1001, productName: 'Mobile', productCategory: 'Gadget'};  
const productDetails: string = getProductDetails(prodObject);  
console.log(productDetails);
```

Output:

```
C:\Users\admin\Pictures>ts-node exp10c.ts  
The product name is : Mobile
```

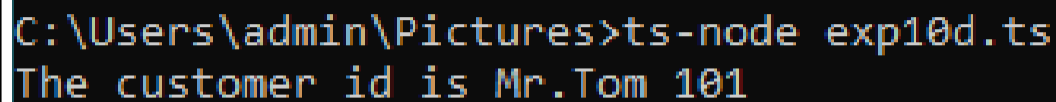
10.d Course Name: Typescript

Module Name: Function Types Declare an interface with function type and access its value

Program:

```
function CreateCustomerID(name: string, id: number): string {  
    return 'The customer id is ' + name + ' ' + id;  
}  
  
interface StringGenerator {  
    (chars: string, nums: number): string;  
}  
  
let idGenerator: StringGenerator;  
idGenerator = CreateCustomerID;  
  
const customerId: string = idGenerator('Mr.Tom', 101);  
console.log(customerId);
```

Output:



```
C:\Users\admin\Pictures>ts-node exp10d.ts  
The customer id is Mr.Tom 101
```

ENLIGHTENS THE NESCIENCE