MCA Department

MCAL14
WEB TECHNOLOGIES LAB

1 Create an application to demonstrate Node.js Modules

Demonstrating Node.js Modules

1. Create a Node.js Application in VS Code:

- Initialize a new Node.js project in VS Code by creating a project folder and running npm init -y to generate a package.json file. This sets up the environment for managing dependencies and project metadata.

2. Demonstrate Local Modules:

- Local modules are custom modules created within the same project. Create a separate JavaScript file to define functions or variables, then export them to be used in other files. Import the module using require in the main application file to demonstrate its usage.

3. Demonstrate Global Module:

- Global modules are built-in modules provided by Node.js that do not require installation. An example is the console module, which allows you to log messages to the console. Simply use the global module directly in your code.

4. Demonstrate Non-Global Core Module:

- Non-global core modules are standard modules provided by Node.js that need to be imported using require. An example is the fs (file system) module, which provides functions to interact with the file system, such as reading or writing files.

5. Demonstrate Third-Party Module:

- Third-party modules are packages developed by the Node.js community, available for installation via npm (Node Package Manager). Install a module like lodash using npm install lodash, then import and use it in your project to demonstrate its functionality.

6. Create Your Own Module Returning Date and Time:

- Create a custom module in a separate file that returns the current date and time. Export the function and import it in the main application file to use it. This demonstrates how to encapsulate functionality in reusable modules.

AIM: 1 Create a Node.js Application in VS Code

CODE:

Step:1) Type npm init in Shell

It will Create package.jason in the folder

Then Create a New File (helloworld) with Extension.js

Write the Following Code

```
console.log('Hello, World!');
```

OUTPUT:

To Run Node File Type: node helloWorld.js

Hello, World!

AIM: 2. Write a node.js program to demonstrate the concept of Local Modules.

CODE:

#Main.js

```
var lib = require('./modlib');

var add= lib.Addition(2,3);

console.log("Addition = "+add);


var sub= lib.subtract(6,5);

console.log("Subtraction = "+sub);
```

#Modlib.js

```
function Addition(a,b){

return a+b;

}

function subtract(a,b){

return a-b;
```

```
}

module.exports ={Addition,subtract};
```

OUTPUT:

Addition = 5

Subtraction = 1

AIM: 3. Write a node.js program to demonstrate the concept of Global Module

CODE:

```
var fs = require("fs");

var data = fs.readFileSync("data.txt","utf-8");

console.log(data);

console.log("------END OF FILE------");
```

OUTPUT:

PS C:\Harshala\ReactJs\pract> node fpdemo3.js

Wuthering Waves is a story-rich open-world action RPG with a high degree of freedom from Kuro Game. You wake from your slumber as Rover, joined by a vibrant cast of Resonators on a journey to reclaim your lost memories and surmount the Lament.

------END OF FILE------

AIM: 4. Write a node.js program to demonstrate the concept of Non-Global Core Module.

CODE:

```
//Asynchronic

var fs = require("fs");
```

```
fs.readFile("data.txt","utf-8",(err,data)=>{console.log(data)});

console.log("------END OF FILE------");
```

Output:

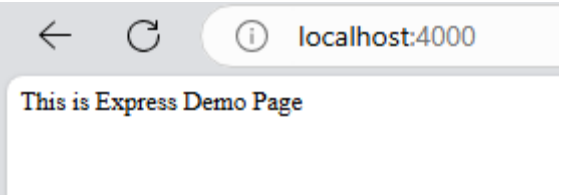
```
PS C:\Harshala\ReactJs\pract> node fpdemo4.js

------END OF FILE------

Wuthering Waves is a story-rich open-world action RPG with a high degree of freedom from Kuro Game.
You wake from your slumber as Rover, joined by a vibrant cast of Resonators on a journey to reclaim
your lost memories and surmount the Lament.
```

AIM : 5. Write a node.js program to demonstrate the concept of Third-Party Module.

CODE:

```
npm install express

const express = require('express');

const app = express();

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

res.send('This is Express Demo Page');

});

app.get('/AboutUs',(req, res)=>{res.send('This is About Us page');});


app.listen(4000);
```

OUTPUT:



MCAL14
WEB TECHNOLOGIES LAB

AIM : 6. Write a program to create your own module that returns date and time object

CODE:

#dateAndTime.js

```
exports.getDateTime = function()
{
    return new Date();
};
```

#app.js

```
const dateTime = require('./dateAndTime');
console.log('Current Date and Time:', dateTime.getDateTime());
```

Run app.js

OUTPUT:

```
PS C:\Harshala\NodeJs\pract1,2,3> node app.js
Current Date and Time: 2024-12-11T07:02:39.563Z
```

MCA Department

MCAL14
WEB TECHNOLOGIES LAB

2 Create an application to demonstrate various Node.js Events

Demonstrating Various Node.js Events

1. **Event Emitter Class Using** on **and** emit**:**
    - This program showcases how to create and manage custom events using the EventEmitter class, registering event listeners with on and triggering them with emit.
2. **Event Emitter Object Using Function:**
    - This example illustrates the use of an EventEmitter object within a function to encapsulate event-driven logic, demonstrating the function's ability to emit and handle events.
3. **Event Emitter Object with Multiple Event Handlers:**
    - This program demonstrates how multiple listeners can be registered for a single event on an EventEmitter object, showing all handlers being executed in response to a single event emission.

AIM: 1. Write a program to demonstrate Event emitter class using properties like on and emit.

CODE:

```
//Example1 On and Emit

//importing event

var events= require('events')

//initializing event emitter instance

var eventEmitter = new events.EventEmitter();

//Call Back Function ()=>

var myEventHandler= ()=>{

console.log("Order Received! Baking a pizza...");

}

//registering event

eventEmitter.on('order-pizza', myEventHandler);

//triggering event

eventEmitter.emit('order-pizza');
```

```
//removing listener
eventEmitter.removeListener('order-pizza',myEventHandler);
console.log(" Listener Removed");
//removing All listener registered to eventEmitter
eventEmitter.removeAllListeners('order-pizza',myEventHandler);
console.log("All listener removed");
```

OUTPUT:

```
Order Received! Baking a pizza...
 Listener Removed
All listener removed
```

AIM: 2. Write a program to demonstrate the concept of event emitter object using function

CODE:

```
//importing event
var events= require('events')
//initializing event emitter instance
var eventEmitter = new events.EventEmitter();
//registering event
eventEmitter.on('order-pizza', (size,topping)=>{
console.log(`Order Received! Baking a ${size} pizza with ${topping}...`);
});
//triggering event
eventEmitter.emit('order-pizza','large','Black Olives');
```

OUTPUT:

PS C:\Harshala\NodeJs\EventHandling> node index1.js

Order Received! Baking a pizza...

AIM: 3. Write a program to demonstrate the concept of event emitter object with multiple event handlers.

CODE:

```
//Example 3(Single Event with multiple event handlers)


//importing event
var events= require('events');
//initializing event
var eventEmitter = new events.EventEmitter();
//registering event
eventEmitter.on('order-pizza'
,(size,topping)=>{
console.log(`Order Received! Baking a ${size} pizza with ${topping}...`);
});
//registering event
eventEmitter.on('order-pizza',(size)=>{
if(size=='large')
{
console.log("Serving Complimentary Sprite bottle!");
}
});
//triggering event
```

```
eventEmitter.emit('order-pizza','large','Black Olives');
```

OUTPUT:

```
PS C:\Harshala\NodeJs\EventHandling> node index3.js

Order Received! Baking a large pizza with Black Olives...

Serving Complimentary Sprite bottle!
```

MCA Department

MCAL14
WEB TECHNOLOGIES LAB

3 Create an application to demonstrate Node.js Functions

Demonstrating Node.js Functions

1. **Named Function:**
    - This program shows how to define and call a named function in Node.js, illustrating the basic syntax and usage.
2. **Function Expressions:**
    - This example demonstrates defining functions as expressions and assigning them to variables, highlighting an alternative way to declare functions.
3. **Arrow Function or Callback Function:**
    - This program illustrates the usage of arrow functions or callback functions to streamline syntax and handle asynchronous operations.
4. **Higher-Order Function:**
    - This example demonstrates higher-order functions, which take other functions as arguments or return them, showcasing the power of functional programming in Node.js.

AIM: 1. Write a node.js program to demonstrate Named function.

CODE:

```
function greet(name)
{
    console.log(`Hello,${name}!`);
}
greet("Harshala");
```

OUTPUT:

```
PS C:\Harshala\NodeJs\FunctionDemo> node index.js
Hello,Harshala!
```

MCAL14
WEB TECHNOLOGIES LAB

AIM: 2. Write a node.js program to demonstrate Function Expressions.

CODE:

```
//Anonymous function Expression
const multiply = function(a,b){
    return a*b;
};
console.log(multiply(5,3));
```

OUTPUT:

```
PS C:\Harshala\NodeJs\FunctionDemo> node index1.js
15
```

AIM: 3. Write a node.js program to demonstrate Arrow Function or callback function.

CODE:

```
//Arrow Function
const sum = (a,b)=>a+b;
console.log(sum(50,5));
```

OUTPUT:

```
PS C:\Harshala\NodeJs\FunctionDemo> node index1.js
55
```

AIM: 4. Write a node.js program to demonstrate Higher order function.

CODE:

```
// higherOrderFunction.js
```

MCAL14
WEB TECHNOLOGIES LAB

```
// Higher-order function that takes a function as an argument
function operate(a, b, operation) {
    return operation(a, b);
}
// Simple functions to be used as callbacks
function add(x, y) {
    return x + y;
}
function subtract(x, y) {
    return x - y;
}

// Using the higher-order function
const num1 = 20;
const num2 = 10;

const sum = operate(num1, num2, add);
const difference = operate(num1, num2, subtract);
console.log(`Sum: ${sum}`);                    // Output: Sum: 30
console.log(`Difference: ${difference}`); // Output: Difference: 10
```

OUTPUT:

```
PS C:\Harshala\NodeJs\FunctionDemo> node highorder.js
Sum: 30
Difference: 10
```

MCA Department

MCAL14
WEB TECHNOLOGIES LAB

4 Using File Handling demonstrate all basic file operations (Create, write, read, delete)

File Handling Demonstrations

1. **Get File Extension:**
    - Demonstrate using path.extname() to extract and display the file extension from a provided file path.
2. **Resolve Paths to Absolute Path:**
    - Use path.resolve() to combine multiple path segments into an absolute path, showcasing its utility in creating standardized paths.
3. **Join Paths:**
    - Illustrate the use of path.join() to concatenate path segments into a single unified path, demonstrating the method's flexibility in path manipulation.
4. **Create Empty File:**
    - Show how to create a new, empty file using the fs.open() method, highlighting its capability to handle file creation and modification flags.
5. **Create and Write to a File:**
    - Utilize the fs.appendFile() method to create a new file or append data to an existing file, ensuring data is added without overwriting.
6. **Read HTML File:**
    - Develop a program that reads an HTML file and outputs its contents, demonstrating file reading capabilities with Node.js.
7. **Read File Synchronously and Asynchronously:**
    - Compare synchronous (fs.readFileSync()) and asynchronous (fs.readFile()) methods for reading file contents, emphasizing performance and blocking behavior.
8. **Delete a File:**
    - Employ the fs.unlink() method to delete a specified file, demonstrating file removal operations within the filesystem.
9. **Rename a File:**
    - Use the fs.rename() method to rename an existing file, showcasing the ability to alter file names and locations within the filesystem.

MCA Department

MCAL14
WEB TECHNOLOGIES LAB

AIM: 1. Write a node.js program to show how to use path.extname () method to get the file extension from a file path

CODE:

```
const path = require("path");

const filepath = ('C:/Users/Administrator/Downloads/Node-practical1.pdf');

const fileExtension =  path.extname(filepath);

console.log('File Extension: '+fileExtension);
```

OUTPUT:

```
PS C:\Harshala\ReactJs\pract> node fpdemo1.js

File Extension: .pdf
```

AIM: 2. Write a node.js program to show how to use the path.resolve () method to resolve a sequence of paths or path segments into an absolute path.

CODE:

```
const path = require("path");

const absolutePath = path.resolve('user','gfg','document');

console.log(absolutePath);
```

OUTOUT:

```
PS C:\Harshala\ReactJs\pract> node fpdemo2.js

C:\Harshala\ReactJs\pract\user\gfg\document
```

AIM: 3. Write a node.js program to demonstrate the use of path.join() method.

CODE:

```
// Import the path module
```

```
const path = require('path');


// Define several path segments
const directory = 'users';
const subDirectory = 'harshala';
const fileName = 'document.txt';


// Use path.join() to combine the path segments into a single path
const filePath = path.join(directory, subDirectory, fileName);


// Output the combined path
console.log('Combined Path:', filePath);
```

OUTPUT:

```
PS C:\Harshala\NodeJs\pract>    node joinpath.js
Combined Path: users\harshala\document.txt
```

AIM: 4. Write a program to create a new empty file using open() method**.**

CODE:

```
// Import the fs module
const fs = require('fs');
const path = require('path');


// Define the path for the new file
const filePath = path.join(__dirname, 'newFile.txt');
```

```
// Use fs.open() to create a new empty file
fs.open(filePath, 'w', (err, file) => {
    if (err) throw err;
    console.log('New empty file created successfully');
});
```

OUTPUT:

```
PS C:\Harshala\NodeJs\pract> node createfile.js
New empty file created successfully
```

AIM: 5. Create a new file using the appendFile() method in node.js.

CODE:

```
// Import the fs module
const fs = require('fs');
const path = require('path');

// Define the path for the new file
const filePath = path.join(__dirname, 'appendFile.txt');
const content = 'This is some content to append to the file.\n';
// Use fs.appendFile() to create a new file and append content
fs.appendFile(filePath, content, (err) => {
    if (err) throw err;
    console.log('File created and content appended successfully');
});
```

OUTPUT:

PS C:\Harshala\NodeJs\pract> node createFileWithAppend.js

File created and content appended successfully

AIM: 6. Create a Node.js file that reads the HTML file, and return the content.

CODE:

#index.html

```
<!DOCTYPE html>
<html>
<head>
   <title>My HTML File</title>
</head>
<body>
   <h1>Hello, World!</h1>
   <p>This is a simple HTML file.</p>
</body>
</html>
```

#server.js

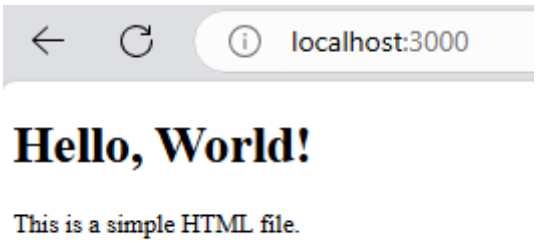
```
const http = require('http');
const fs = require('fs');
const path = require('path');
// Define the path to the HTML file
const filePath = path.join(__dirname, 'index.html');
```

```
// Create an HTTP server
const server = http.createServer((req, res) => {
    fs.readFile(filePath, (err, data) => {
        if (err) {
            res.statusCode = 500;
            res.end('Error reading file');
            return;
        }
        res.statusCode = 200;
        res.end(data);
    });
});

// Define the port and start the server
const port = 3000;
server.listen(port, () => {
    console.log(`Server running at http://localhost:${port);
});
```

OUTPUT:

```
PS C:\Harshala\NodeJs\pract> node server.js
Server running at http://localhost:3000/
```

MCAL14
WEB TECHNOLOGIES LAB



AIM: 7. Write a program to read content of file in node.js synchronously and asynchronously.

CODE:

//Synchronic

```
var fs = require("fs");

var data = fs.readFileSync("data.txt","utf-8");

console.log(data);

console.log("------END OF FILE------");
```

OUTPUT:

```
PS C:\Harshala\ReactJs\pract> node fpdemo3.js

Wuthering Waves is a story-rich open-world action RPG with a high degree of freedom from Kuro Game. You wake from your slumber as Rover, joined by a vibrant cast of Resonators on a journey to reclaim your lost memories and surmount the Lament.

------END OF FILE------
```

//Asynchronic

```
var fs = require("fs");

fs.readFile("data.txt","utf-8",(err,data)=>{console.log(data)});

console.log("------END OF FILE------");
```

Output:

PS C:\Harshala\ReactJs\pract> node fpdemo4.js

------END OF FILE------

Wuthering Waves is a story-rich open-world action RPG with a high degree of freedom from Kuro Game. You wake from your slumber as Rover, joined by a vibrant cast of Resonators on a journey to reclaim your lost memories and surmount the Lament.

AIM: 8. Write a program to delete a file with the File System module (use the fs.unlink() method)

CODE:

```
// Import the File System module

const fs = require('fs');



// Specify the path to the file you want to delete

const filePath = './example.txt';



// Use the fs.unlink() method to delete the file

fs.unlink(filePath, (err) => {

   if (err) {

      console.error(`Error deleting file: ${err.message}`);

      return;

   }

   console.log('File deleted successfully');

});
```

OUTPUT:

S:\MCA\NodeJs>node index.js

File deleted successfully

AIM: 9. Write a program to rename a file with the File System.

CODE:

```
//First Create a file with name "oldFileName.txt"
// Import the fs module
const fs = require('fs');
const path = require('path');

// Define the path to the original file and the new file name
const oldPath = path.join(__dirname, 'oldFileName.txt');
const newPath = path.join(__dirname, 'newFileName.txt');
// Use fs.rename() to rename the file
fs.rename(oldPath, newPath, (err) => {
    if (err) throw err;
    console.log('File renamed successfully');
});
```

OUTPUT:

```
PS C:\Harshala\NodeJs\pract> node renamefile.js
File renamed successfully
```

MCAL14
WEB TECHNOLOGIES LAB

5 Create an HTTP Server and perform operations on it

Demonstrating HTTP Server Operations in Node.js

1. **Create an HTTP Server:**
    - This program creates an HTTP server that listens on a specified port and sends a response back to the client using the http module in Node.js.
2. **Set Content-Type of HTTP Response:**
    - This example demonstrates setting the Content-Type header in the HTTP response to inform the client about the type of content being sent, ensuring proper handling and display.
3. **Shut Down the Running HTTP Server:**
    - This program illustrates how to gracefully shut down a running HTTP server, ensuring that all ongoing requests are completed before the server is terminated.

AIM: 1. Write a program to create HTTP Server that listens to server ports and gives a response back to the client using http module.

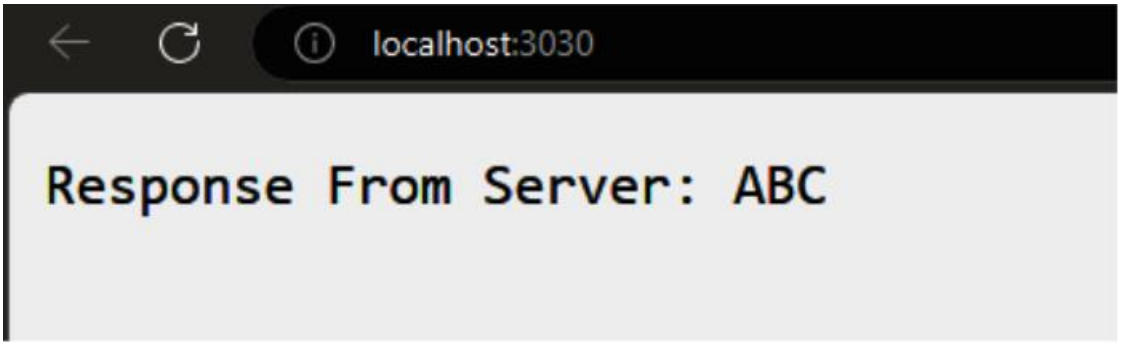
CODE:

```
//Index.js

//Create HTTP server that listen to server port


var http = require('http');

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

res.write('Response From Server: ABC');

res.end();

});

server.listen(3030,'localhost',()=>{

    console.log("Server running at http://localhost:3030");

});
```

OUTPUT:

```
PS C:\Harshala\NodeJs\HTTPServer> node index.js
```

Server running at http://localhost:3030



AIM2. Write a program to set content-type of HTTP Response.

CODE:

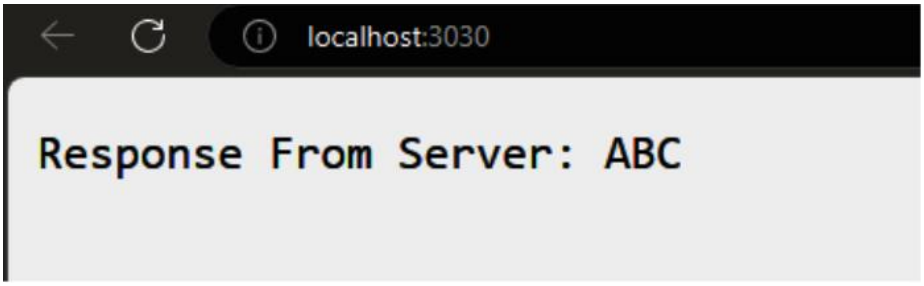
//Index1.js

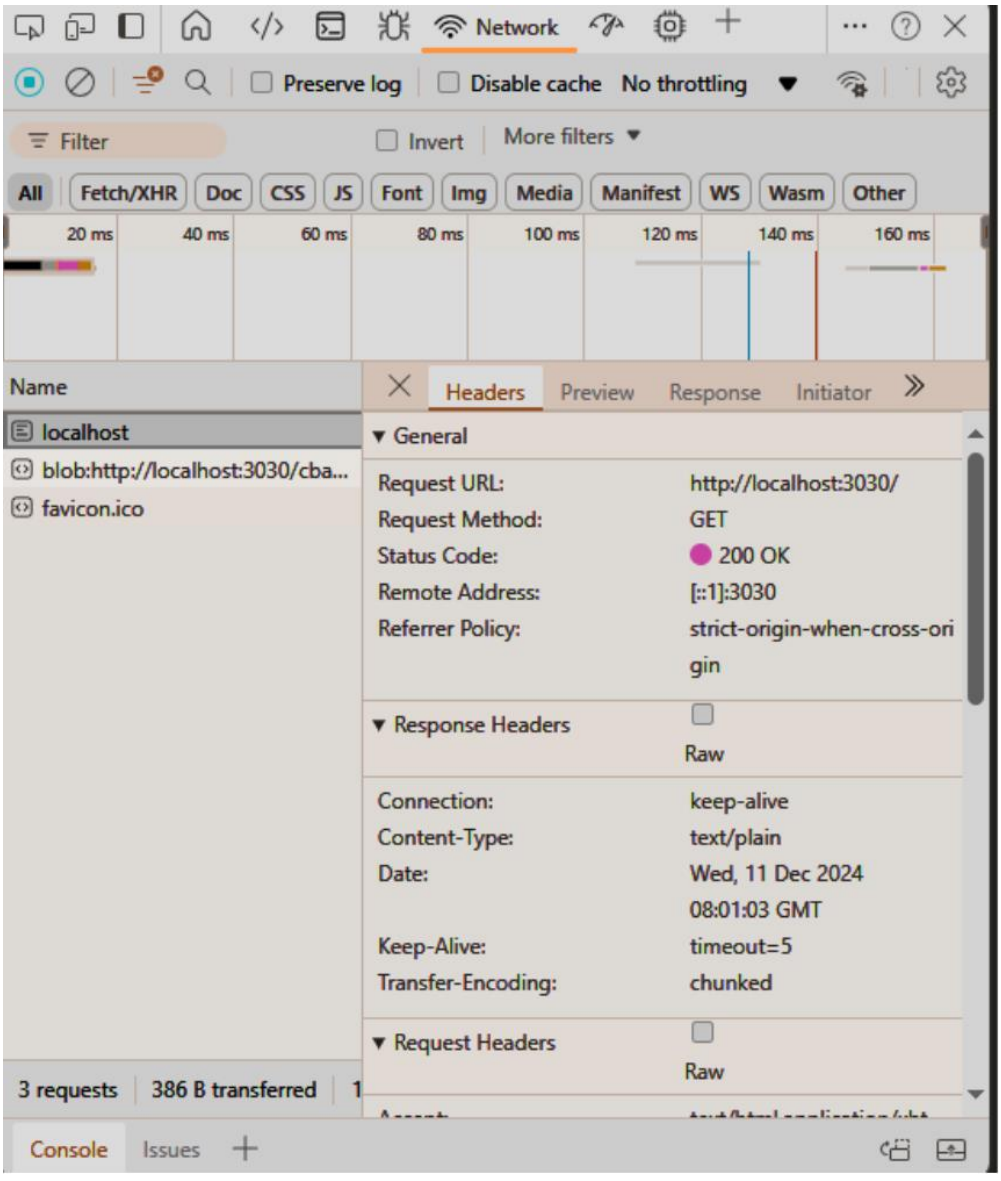
```
var http = require('http');

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

    {

        res.setHeader('Content-Type','text/plain');

        res.write('Response From Server: ABC');

        res.end();    });

    server.listen(3030,'localhost',()=>

    {

        console.log("Server running at http://localhost:3030");

    });
```

OUTPUT:

```
PS C:\Harshala\NodeJs\HTTPServer> node index1.js

Server running at http://localhost:3030
```

MCAL14
WEB TECHNOLOGIES LAB





AIM2. 3. Write a program to shut down the Running HTTP Server

CODE:

```
//Index1.js


var http = require('http');
const server = http.createServer((req, res)=>{
        res.setHeader('Content-Type','text/plain');
        res.write('Response From Server:');
        res.end();
});
server.listen(3030,'localhost',()=>{
        console.log("Server running at http://localhost:3030");
server.close(()=>{
        console.log("Server Closed");
        });
});
```

OUTPUT:

```
PS C:\Harshala\NodeJs\HTTPServer> node index2.js
Server running at http://localhost:3030
Server Closed
```

MCA Department

MCAL14
WEB TECHNOLOGIES LAB

6 Create an application to establish a connection with the MySQL database and perform basic database operations on it.

Establishing a Connection with MySQL and Performing Basic Database Operations

1. **Connect with MySQL Database:**
    - This program demonstrates how to establish a connection to a MySQL database using Node.js and the mysql module.
2. **Select Table Data:**
    - This example shows how to retrieve and display all records from a specified table in the MySQL database.
3. **Select One Record:**
    - This program retrieves a single record from a MySQL database table, demonstrating how to query for specific data.
4. **Select Employee Name by ID:**
    - This example selects and displays the name of an employee with a specific ID (id=1) from the MySQL database.
5. **Create a Table 'emp1':**
    - This program shows how to create a new table named emp1 in the MySQL database using a Node.js script.
6. **Insert a Record in 'emp1' Table:**
    - This example demonstrates how to insert a new record into the emp1 table, highlighting basic data insertion operations.

AIM: 1. Write node.js program to connect with MySQL Database.

CODE:

npm init

npm install mysql

```
var mysql = require('mysql');

var conn = mysql.createConnection({
```

```
    host: 'localhost',
    user:'root',
    password: 'root123',                    //new password
    database:'MCA'
});

conn.connect(function(error)
{
if(error) throw error;
console.log("Connected");
}
);
```

#Create SQL_Script file

```
Create database MCA;
Use MCA;
Create table emp (id int, ename varchar(20), age int);
Insert into emp values(1,'ABC',45);
Insert into emp values(2,'XYZ',34);
Insert into emp values (3,'PQR', 32);
Select * from emp;

show variables like 'default_authentication_plugin';
select user, plugin
FROM mysql.user
```

```
where user='root';


#Have to run from here again

alter user 'root'@'localhost' identified with

mysql_native_password by 'root123';


flush privileges;
```

OUTPUT:

```
PS C:\Harshala\NodeJs\DatabaseDemo> node index.js

Connected
```

AIM: 2. Write node.js program to select table data from MySQL database.

CODE:

```
var mysql = require('mysql');

var conn = mysql.createConnection({

      host: 'localhost',

      user:'root',

      password: 'root123',

      database:'MCA'

});

conn.connect( function(error)

{

   if(error) throw error;

     conn.query("select * from emp",function(error,result){
```

```
    if(error) throw error;
      console.log(result);
    });
});
```

OUTPUT:

```
PS C:\Harshala\NodeJs\DatabaseDemo> node index2.js
[
    RowDataPacket { id: 1, ename: 'ABC', age: 45 },
    RowDataPacket { id: 2, ename: 'XYZ', age: 34 },
    RowDataPacket { id: 3, ename: 'PQR', age: 32 }
]
```

AIM: 3. Write node.js program to select only one record from MySQL database.

CODE:

```
var mysql = require('mysql');
var conn = mysql.createConnection({
        host: 'localhost',
        user:'root',
        password: 'root123',
        database:'MCA'
});
conn.connect( function(error)
{
    if(error) throw error;
    conn.query("select * from emp",function(error,result)
```

```
   {
   if(error) throw error;
   console.log(result[0]);
   });
});
```

OUTPUT:

```
PS C:\Harshala\NodeJs\DatabaseDemo> node index3.js
RowDataPacket { id: 1, ename: 'ABC', age: 45 }
```

AIM: 4. Write node.js program to select name of employee having id=1from MySQL database..

CODE:

```
var mysql = require('mysql');
var conn = mysql.createConnection({
        host: 'localhost',
        user:'root',
        password: 'root123',
        database:'MCA'
});
conn.connect( function(error)
{
   if(error) throw error;
   conn.query("select * from emp",function(error,result){
   if(error) throw error;
   console.log(result[0].ename);
   });
```

```
});
```

OUTPUT:

```
PS C:\Harshala\NodeJs\DatabaseDemo> node index4.js
ABC
```

AIM: 5. Write node.js program to create a table ‘emp1’.

CODE:

```
var mysql = require('mysql');
var conn = mysql.createConnection({
      host: 'localhost',
      user:'root',
      password: 'root123',
      database:'MCA'
});
conn.connect( function(error)
{
   if(error) throw error;
   conn.query("create table emp2(id int, ename varchar(10))",function(error){
   if(error) throw error;
   console.log("Table created");
   });
});
```

OUTPUT:

```
PS C:\Harshala\NodeJs\DatabaseDemo> node index5.js
Table created
```

MCAL14
WEB TECHNOLOGIES LAB

AIM: 6. Write node.js program to insert a record in emp1 table

CODE:

```
//Index5.js


var mysql = require('mysql');
var conn = mysql.createConnection({
	host: 'localhost',
	user:'root',
	password: 'root123',
	database:'MCA'
});
conn.connect( function(error)
{
	if(error) throw error;
	conn.query("insert into emp1 values(1,'KF')",function(error){
		if(error) throw error;
		console.log("Record inserted");
	});
});
```

OUTPUT:

```
PS C:\Harshala\NodeJs\DatabaseDemo> node index6.js
Record inserted
```

MCAL14
WEB TECHNOLOGIES LAB

7 Create an application in React JS to implement component life cycle.

Implementing Component Lifecycle in React JS

Develop a React JS application that demonstrates the lifecycle methods of a component, including componentDidMount, componentDidUpdate, and componentWillUnmount, to handle various stages of the component's existence. This helps manage state, side effects, and cleanup tasks effectively within React components.

AIM: Create an application in React JS to implement component life cycle

CODE:

#create index.js in React folder

```
import React from "react";

import ReactDOM from "react-dom/client";

class Test extends React.Component {

 constructor(props) {

 super(props);

 this.state = { hello: ", Hello MCA Students!" };

 }

 componentDidMount() {

 console.log("componentDidMount()");

 }

 changeState()

 {

     this.setState({ hello: " This is a great React Tutorial!" });

 }

 render()

 {

     return (
```

```
      <div>
            <h1>
                  React JS Component Life Cycle{this.state.hello}
            </h1>
            <h2>
                  <a onClick={this.changeState.bind(this)}>
                        Press Here!
                  </a>
            </h2>
      </div>
      );
 }

 shouldComponentUpdate(nextProps, nextState) {
 console.log("shouldComponentUpdate()");
 return true;
}
componentDidUpdate() {
 console.log("componentDidUpdate()");
}
}
const root = ReactDOM.createRoot(
document.getElementById("root")
);
root.render(<Test />);
```

MCAL14
WEB TECHNOLOGIES LAB

OUTPUT:

MCA Department

MCAL14
WEB TECHNOLOGIES LAB

8 Create an application to implement class and functional component in React JS.

Implementing Class and Functional Components in React JS

1. **Class Component:**
    - This program demonstrates how to create and use a class component in React, including defining state and lifecycle methods to manage component behavior.
2. **Functional Component:**
    - This example showcases the creation and usage of a functional component in React, utilizing hooks like useState and useEffect to handle state and side effects.

AIM: 1. Write a node.js program to implement class component

CODE:

#Create New File Instruction.js in React Folder

```
import React, { Component } from 'react'


export default class Instructions extends Component {
    render() {
        return (
            <div className='container'>
            <h1>Assassins Creed</h1>
            <p><h2>BrotherHood</h2></p>
            <img src='./image/a.jpg'></img>
            </div>
        )
    }
}
```

#Now make changes in App.css

.container{

display : flex;

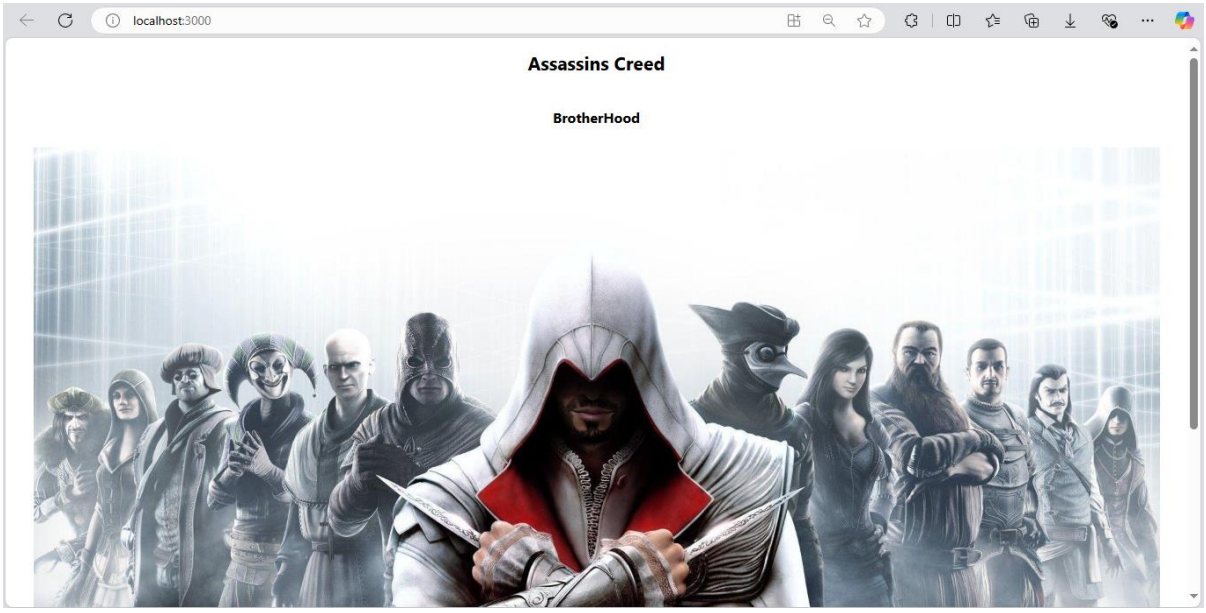
flex-direction: column

align-items: center;

}

Create one folder as ‘image’ in current project in public folder> paste the image p1.jpg

OUTPUT:



AIM: 2. Write a node.js program to implement functional component.

CODE:

#Make Changes in App.js

import logo from './logo.svg';

import './App.css';

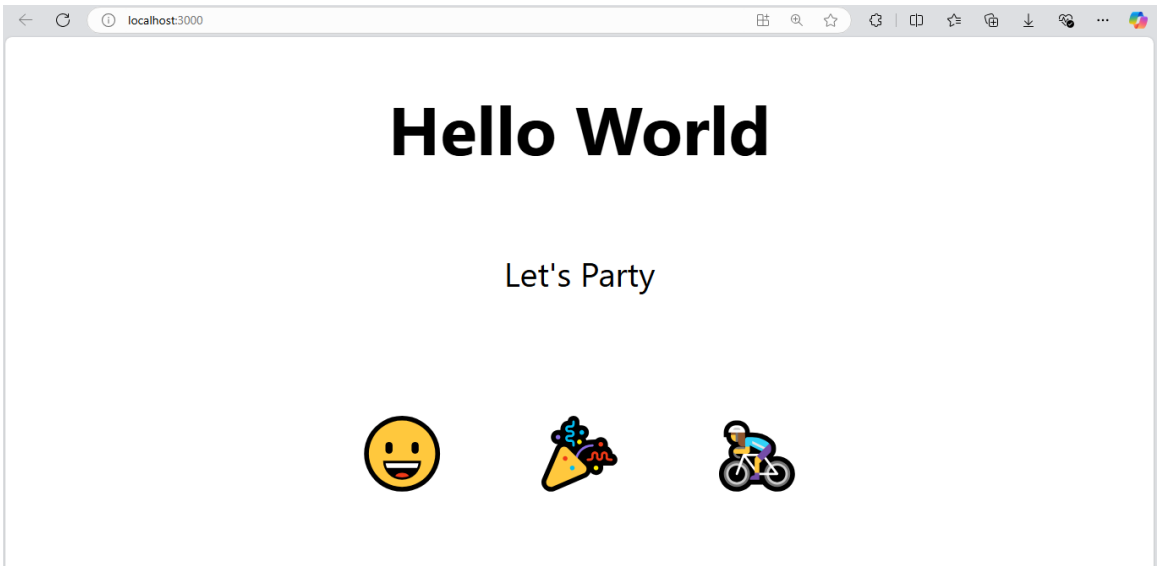
```
function App() {
   return (
      <div className='App'>
         <h1>Hello World</h1>
         <p>Let's Party</p>
         <ul>
            <li><button><span role='img' aria-label='grinning face' id='grinning face'>😀 </span></button></li>
            <li><button><span role='img' aria-label='party popper' id='party popper'>🎉 </span></button></li>
            <li><button><span role='img' aria-label='women dancing' id='women dancing'> 🚴
</span></button></li>
         </ul>
      </div>
   );
}
export default App;
```

#Make Changes in App.css

```
.container{
  display: flex;
flex-direction: column;
align-items: center;
}
button{
font-size: 2em;
border: 0;
```

```
padding: 0;

background: none;

cursor: pointer;

}

ul{

display: flex;

padding: 0;

}

li{

margin: 20px;

list-style: none;

padding: 0;

}
```

OUTPUT:



MCA Department

MCAL14
WEB TECHNOLOGIES LAB

9 Create an application in React JS to import and export the files (components)

Importing and Exporting Files in React JS

Develop a React JS application to demonstrate the import and export functionality, illustrating how to organize components in separate files, export them, and import them into the main application or other components for better modularity and code organization.

AIM: Create an application in React JS to import and export the files (components)

CODE:

#Make Changes in App.js

```
// src/App.js

import React from 'react';

import Header from './Header';

import Footer from './Footer';


function App() {
   return (
      <div className="App">
         <Header />
         <main>
            <p>This is the main content of the app.</p>
         </main>
         <Footer />
      </div>
   );
}

export default App;
```

MCAL14
WEB TECHNOLOGIES LAB

#Make Header.js

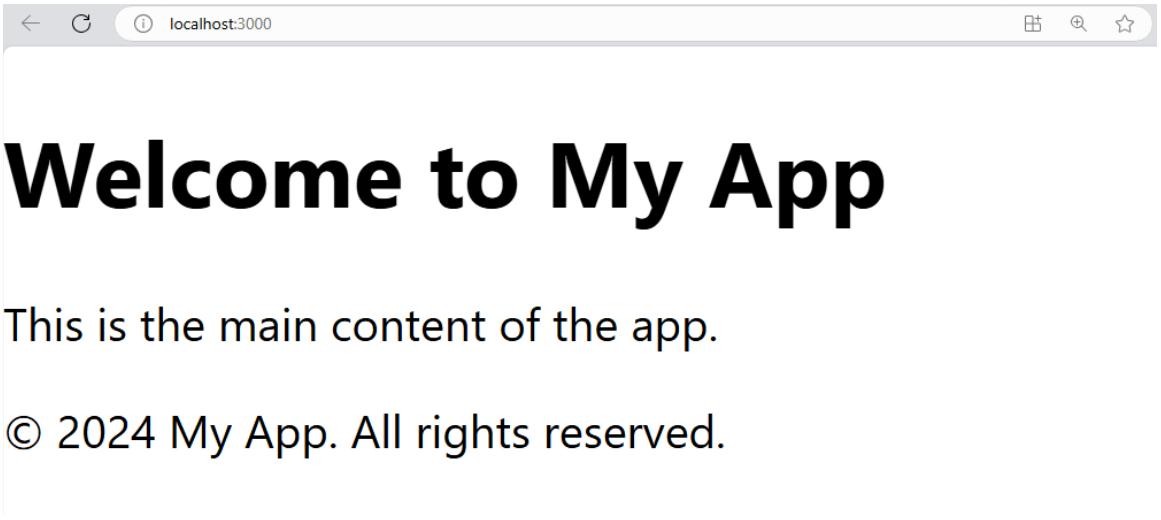
```
// src/Header.js
import React from 'react';


const Header = () => {
   return (
      <header>
         <h1>Welcome to My App</h1>
      </header>
   );
};
export default Header;
```

#Make Footer.js

```
// src/Footer.js
import React from 'react';
const Footer = () => {
   return (
      <footer>
         <p>&copy; 2024 My App. All rights reserved.</p>
      </footer>
   );
};
export default Footer;
```

MCAL14
WEB TECHNOLOGIES LAB

OUTPUT:



MCAL14
WEB TECHNOLOGIES LAB

10 Create an application to implement state and props

Implementing State and Props in React JS

Develop a React JS application that demonstrates the use of state within components to manage dynamic data and props to pass data between components, enabling efficient data handling and communication within the application.

AIM: Create an application to implement state and props

CODE:

#Create Product.js

```
import React, { Component } from 'react';

import './Product.css';


const products = [
  {
    pr: '🍦',
    name: 'ice cream',
    price: 50
  },
  {
    pr: '🍩',
    name: 'donuts',
    price: 190,
  },
  {
    pr: '🍉',
    name: 'watermelon',
    price: 30
```

```
  }
];

class Product extends Component {
  state = {
    cart: [],
    total: 0
  }

  currencyOptions = {
    minimumFractionDigits: 2,
    maximumFractionDigits: 2,
  }

  getTotal = () => {
    return this.state.total.toLocaleString(undefined, this.currencyOptions);
  }

  add = (product) => {
    this.setState(state => ({
      cart: [...state.cart, product.name],
      total: state.total + product.price
    }));
  }
```

```
remove = (product) => {
    this.setState(state => {
        const cart = [...state.cart];
        const index = cart.indexOf(product.name);
        if (index >= 0) {
            cart.splice(index, 1);
            return {
                cart,
                total: state.total - product.price
            };
        }
        return state;
    });
}

render() {
    return (
        <div className="wrapper">
            <div>
                Shopping Cart: {this.state.cart.length} items
            </div>
            <div>Total: {this.getTotal()}</div>
            <div>
                {products.map(product => (
                    <div key={product.name}>
```

```
                    <div className="product">
                       <span role="img" aria-label={product.name}>{product.pr}</span>
                    </div>
                    <button onClick={() => this.add(product)}>Add</button>
                    <button onClick={() => this.remove(product)}>Remove</button>
                  </div>
                ))}
              </div>
            </div>
        );
    }
}
export default Product;
```

#Create Product.css

```
.product span {
      font-size: 100px;
 }
 .wrapper {
      padding: 20px;
      font-size: 20px;
      flex: auto;
 }
 .wrapper button {
      font-size: 20px;
```

```
    background: blueviolet;
}
```

#Make Changes in App.js

```
import './App.css';
import Product from './Product';
function App() {
  return (
    <>
    <Product></Product>
    </>
  );
}
export default App;
```

#Import Product.js as given below in Index.js

```
import Product from './Product';
```

MCAL14
WEB TECHNOLOGIES LAB

OUTPUT:

Shopping Cart: 3 items
Total: 270.00

Add Remove

Add Remove

Add Remove

MCAL14
WEB TECHNOLOGIES LAB

11 Create an application in React JS to use DOM events

Using DOM Events in React JS

Develop a React JS application that demonstrates handling various DOM events, such as click, change, and mouseover, within components to create interactive and dynamic user experiences. This highlights the integration of event listeners and handlers in a React environment.

AIM: Create an application in React JS to use DOM events

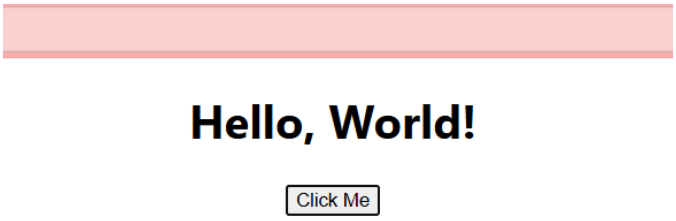
CODE:

```
import React, { Component } from 'react';

import './App.css';


class App extends Component {

  state = {

    text: 'Hello, World!'

  };


  handleClick = () => {

    this.setState({

      text: 'You clicked the button!'

    });

  };


  handleMouseOver = () => {

    this.setState({

      text: 'Mouse is over the button!'

    });

  };
```

```
  handleMouseOut = () => {
    this.setState({
      text: 'Hello, World!'
    });
  };

  render() {
    return (
      <div className="App">
        <h1>{this.state.text}</h1>
        <button
          onClick={this.handleClick}
          onMouseOver={this.handleMouseOver}
          onMouseOut={this.handleMouseOut}
        >
          Click Me
        </button>
      </div>
    );
  }
}

export default App;
```

MCAL14
WEB TECHNOLOGIES LAB

OUTPUT:



MCAL14
WEB TECHNOLOGIES LAB

12 Create an application to implement React Hooks

Implementing React Hooks

Develop a React JS application that demonstrates the use of React Hooks, such as useState for state management and useEffect for side effects, enabling functional components to handle stateful logic effectively.

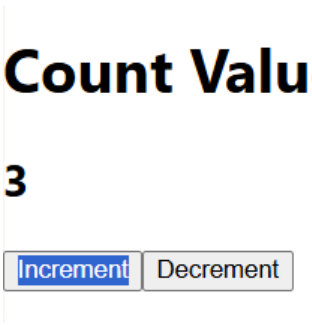
AIM: 1. useState

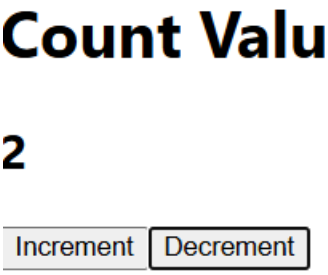
CODE:

```
import React, { useState } from "react";

function App() {

 const [count, setCount] = useState(0);

 const increment = () => setCount(count + 1);

 const decrement = () => {

       if(count>0){

          setCount(count - 1);

       }

 }

 return (

    <div>

       <div>

          <h1 >Count Values</h1>

          <h2>{count}</h2>

          <div>

             <button onClick={increment}>

                Increment

             </button>

             <button onClick={decrement}>

                Decrement
```

```
            </button>
          </div>
        </div>
      </div>
  );
}
export default App;
```

OUTPUT:





MCA Department

MCAL14
WEB TECHNOLOGIES LAB

AIM: 2. useEffect

CODE:

```
import { useState, useEffect } from "react";
 import ReactDOM from "react-dom/client";
 function App() {
  const [count, setCount] = useState(0);
  useEffect(() => {
     setTimeout(() => {
        setCount((count) => count + 1);
     }, 1000);
  });
  return <h1>Your Timer is On: {count} second</h1>;
 }
 const root = ReactDOM.createRoot(document.getElementById('root'));
 root.render(<App />);
 export default App;
```

OUTPUT:

MCA Department

MCAL14
WEB TECHNOLOGIES LAB

13 Create SPA using React Router

Creating an SPA Using React Router

Develop a Single Page Application (SPA) in React using React Router to manage navigation between different views or components without reloading the page, ensuring seamless and efficient user experience. This demonstrates how to define routes, link to different components, and handle client-side routing effectively.

AIM: 1. useState

CODE:

#npm install react-router-dom

#Make Changes in App.js

```
import React from "react";
import './App.css';
import { BrowserRouter, Route, Routes, Link } from "react-router-dom";
import Manatee from "./Manatee";
import Narwhal from "./Narwhal";
import Whale from "./Whale";

function App() {
  return (
    <div>
      <h1>Marine Mammals</h1>
      <BrowserRouter>
        <nav>
          <ul>
            <li><Link to="/manatee">Manatee</Link></li>
            <li><Link to="/narwhal">Narwhal</Link></li>
```

```
                <li><Link to="/whale">Whale</Link></li>
              </ul>
            </nav>
            <Routes>
              <Route path="/manatee" element={<Manatee />} />
              <Route path="/narwhal" element={<Narwhal />} />
              <Route path="/whale" element={<Whale />} />
            </Routes>
          </BrowserRouter>
        </div>
    );
}

export default App;
```

#Create Following Files

#Manatee.js

```
import React from "react";
export default function Manatee(){
    return <h2>Manatee</h2>;
}
```

 # Narwhal.js

```
import React from "react";
export default function Narwhal(){
```

```
    return <h2>Narwhal</h2>;
}
```

#Whale.js

```
import React from "react";
export default function Whale(){
return(
    <h2>Whale</h2>
);}
```

OUTPUT:

Marine Mammals

- Manatee
- Narwhal
- Whale

Manatee

Marine Mammals

- Manatee
- Narwhal
- Whale

Narwhal

Marine Mammals

- Manatee
- Narwhal
- Whale

Whale