Name : Vishal Gupta
Class : TE-IT
Roll No : 13

Internet Programming
Experiment - 1

Aim: Create a webpage using various HTML tags

Theory: Hypertext is text displayed on a computer or other electronic device with references to other text that the user can immediately access, usually by a mouse click or key press.

Code:

```
<!DOCTYPE html>
<html>
<head>
<title> Document </title>
</head>
<body>
  <h1> Hiii </h1>
  <p> Im Vishal </p>
  <ul> hobbies </ul>
  <li> Gaming </li>
  <li> drawing </li>
</body>
</html>
```

Output:

Hiii
Im Vishal
Hobbies
gaming
drawing

Name: Vishal Gupta
Class : TE-IT
Roll No: 13

Experiment-2

Aim: Define Styles for your webpage using CSS

Theory: CSS is designed to make Style sheets for the web. It is independent of HTML and can be used with any XML-based markup language. It is the language for describing the presentation of Webpages, including colours, layout and fonts, thus making our webpage presentable to the users.

Code:

```
<!DOCTYPE html>
<html>
<head>
<title> Exp2 </title>
</head>
<body>
<div class="container">
<h1> Vishal </h1>
<h2> Gupta </h2>
</body>
</html>
```

Styles.css

```
body {
    c
```

Styles.css

```
.container{
    display : flex;
    flex : 6
}

h1{
    text-decoration : underline
}
```

Output :

Vishal Gupta.

Experiment - 3

Aim: Create responsive webpage using Bootstrap

Theory: Bootstrap is a popular front-end framework for web development. It contains pre-built components and design elements to style HTML content.
Bootstrap includes a responsive grid system for varying layouts. It is a great starting point for building a mobile friendly website

Code:

```
<html>
<head>
<link href="Bootstrap link" ></link>
<title> Bootstrap </title>
</head>
<body>
 <div class = "container">
  <div class = "row">
   <div class = "col-sm">
    One of three columns
   </div>
  <div class = "col-sm">
   Two Second of three columns
   </div>
  <div class = "col-sm">
   Third column
   </div>
  </div>
 </div>
</body>
```

</html>

Output:

One of three column Second of three column Third column

Experiment-4

Aim: Create interactive website using javaScript

Theory: JavaScript is a programming language we can use to make a website interactive. When we search something on google or click on link, our website changes - that's what JavaScript allows us to do.

Code:

```
<html>
<body>
<h1> the Onclick Event </h1>
<button id="demo" onclick="myfunction()">Click me </button>
<script>
function myfunction(){
  document.getElementById("demo").innerHTML=
        "Hello world";
}
</script>
</body>
</html>
```

Output :

The Onclick Event

[Click me]

// After clicking button

[Hello world.] // It will change button text

Experiment -5

Aim: Implement arrow function in JavaScript

Theory: Arrow function introduced in ES6, provides a concise way to write functions in javascript. In other words, the context inside arrow functions is lexically or statically defined.

Code:

```
<html>
<body>
<h1> Arrow function </h1>
<p> This is example shows an Arrow function </p>
<p id="demo"></p>
<script>
let hello = " ";
hello=() => "Hello World!";
document.getElementById("demo").innerHTML=hello();
<script>
</body>
</html>
```

Output:

The Arrow function

This is example shows an Arrow function

Hello World!

Experiment-6

Aim:- Implement classes and inheritance in JavaScript

Theory: The JavaScript inheritance is a mechanism that allows us to create new classes on the basis of already existing classes. It provides flexibility to the child class to reuse the methods and variables of a parent class. The javascript extends keyword is used to create a child class on the basis of a parent class.

Code:

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
 class Car{
  constructor(brand){
   this.carname = brand;
  }
  present(){
   return 'I have a' + this.carname;
  }
}
```

```
class Model extends Car {
  constructor(brand, mod) {
    super(brand);
    this.model = mod;
  }
  show() {
    return this.present() + ', it is a ' + this.model;
  }
}
let myCar = new Model("Ford", "Mustang");
document.getElementById("demo").innerHTML = myCar.sho
</script>
</body>
</html>
```

Output:

I have a Ford, it is a Mustang.

Experiment - 7

Aim:- Implement fetch function of JavaScript.

Theory:- The fetch() method in JavaScript is used to request to the server and load the information in the webpages. The request can be of any APIs that returns the data of the format JSON or XML. This method returns a promise

Code:

```
<!DOCTYPE html>
<html>
<body>
<h1> fetch in Javascript </h1>
<script>
 let fetchRes = fetch("https://jsonplaceholder.typicode.com/todo
fetchRes.then(res => res.json()).then(d => console.log(d)
</script>
</body>
</html>
```

Output:

Output :

Document

fetch() in Javascript

Console

{
"userId" : 1,
"Id" : 1;
"title" : "delectus aut autem"
"completed" : false
}

Experiment-8

Aim:- Implement State and Props in React.

Theory: State: ☆ State is a variable which exist inside a component, that cannot be accessed and modified outside the component.

Props: Sometime we need to change the context inside a component. Props come to play in these cases, as they are passed into the component.

Code:

```
//Props
Class Car extends React.Component{
  constructor(props){
    super(props);
    this.state ={
      brand : "Ford";
      Model : "Mustang"
    };
  }
  render(){
    return(
    <div>
    <h1> My{this.state.brand}</h1>
    </div>
    );
  }
}
```

Output:

Ny Ford

Experiment - 9

Aim: Implement React Router.

Theory: React Router is a standard library for routing in React. It enables the navigation among views of various components in a React Application, allows changing the browser URL, and keeps the UI in sync with the URL. Lets us create a simple application to React to understand how the React Router works.

Code:

Layout.js:

```
import {Outlet, Link} from "react-router-dom";
const layout = ()=>{
return(
  <>
  <nav>
   <li>
    <Link to="/">Home </Link>
   </li>
   <li>
    <Link to="/blogs"> Blogs </Link>
   </li>
  </ul>
  </nav>
  <Outlet>
  </>
 )
};
```

```
export default Layout;
```

Home.js

```
const Home = () => {
  return <h1>Home</h1>;
};
export default Home;
```

Blogs.js

```
const Blogs = () => {
  return <h1> Blog </h1>;
};
export default Blogs.
```

index.js

```
import ReactDOM from "react-dom/client";
import { BrowserRouter, Route } from "react-router";
import Layout from "./Layout";
import Home from "./Home";
import Blogs from "./Blogs";

export default function App() {
  return (
    <BrowserRouter>
    <Routes>
      <Route path="/" element={<Layout />}>
      <Route path="Home" element={<Home />} />
      <Route path="Blog" element={<Blog />} />
    </Routes>
  </BrowserRouter>
  );
}
```

Output :

localhost : 3000 / home

Home

localhost : 3000 / Blog

Blog.

Experiment - 10

Aim: Implement React forms

Theory: Forms are an integral part of any modern web application. It allows the users to interact with the application as well as gather information from the users. Forms can perform many tasks that depends on the nature of your business requirements and logic such as authentication of the user, adding a user, searching, filtering, booking, ordering, etc.

Code:

```
import {useState} from "react";
import React
function myform(){
  const [name, setName] = useState("");
  return(
   <form>
    <label> Enter your name:
     <input
       type="text"
       value={name}
       onChange={(e)=>setName(e.target.value)}
      />
    </label>
    </form>
  )
}
```

Output :

Enter your name :

Experiment - 11

Aim:- Implement styles using ReactStrap

Theory:- ReactStrap is a popular front-end library that is easy to use React Bootstrap 4 components. This library contains the Stateless React components for Bootstrap 4.

Code:-

```
import Button from 'react-bootstrap/Button';

function StyleExample (){
  return (
    <>
    <Button variant="outline-primary">Primary</Button>
    <Button variant="outline-dark">Dark</Button>
    <>
  );
}
export default StyleExample;
```

Output:

(Primary) (Dark)

Experiment - 12

Aim: Implement useEffects and useState in React

Theory:- A React HOOK is a new feature to use the React component features in a functional component.
The useState() hook for using States from function components.
The useEffect() hook for performing side effects from function components

Code:-

```
import {use State, useEffect} from "react";
import
function Timer(){
  const [count, setCount] = useState(0);
  useEffect(()=>{
    setTimeout(()=>{
      setCount((count)=>count+1);
    },1000);
  });
  return <h1>I have rendered {count} times!</h1
}
```

Output:

I have rendered 1 time!

// After 1 second

I have rendered 2 time!

.

.

.

Experiment - 13

Aim: Implement Ref in React.

Theory:- Refs are a function provided by React to access the DOM elements and the React element that you might have created on your own. They are used in cases where we want to change the value of a child component, without making use of props and all. They also provide us with good functionality as we can use callbacks with them.

Code:

```
const Counter = () => {
  const prevCount = useRef(0);
  const [counter, setCounter] = useState(0);
  useEffect(() => {
    console.log('counter:', counter, 'prevCounter:', preCou
  }, [prevCount, counter]);

  return (
  <div className="App">
    <p>{counter}</p>
    <button
      onClick={() => {
```

```
setCounter(counter => {
  prevCount.current = counter;
  return counter + 1;
});
}}
>
Increment Counter
</button>
</div>
);
};
```

Output:

Console:

counter : 1	prevCount : 0
counter : 2	prevcount : 1
counter : 3	prevcount : 2

Experiment-14

Aim: Implement custom HOOK in React

Theory: ☆ custom HOOK is a JavaScript function. The name of custom HOOK starts with "use" which can call other hooks. ☆ custom HOOK is just like a regular function, and the word "use" in the begining tells that this function follows the rules of HOOKS. Building custom HOOKS allows you to extract component logic into reuseable function

Code:

```
import React, {useState, useEffect} from "React";
function customHook<(data){
  useeffect (()=>{
   document.title = data;
  }
  [data]);
  Return data
}
function HOOK(){
 const SuperValue = customHook ("Vidyavardhini")
 Return (<p>The new value is {SuperValue} </p>);
}
export default HOOK;
```

Output:

localHost:3000

The new value is Vidyavardhini

Experiment - 15

Aim: Bundling React app using Webpack

Theory: Webpack, a static module bundler, a complex yet simple tool that allows you to spend between 10 minutes and 10 hours to make a simple web application bundlable. Webpack delivers a bundle of code that's easily parseable for your browser, or node environment.

Code:

```
impost {add} from "/math.js";
bundled code instead of imports
function app (){
 function add (a,b){
  return a+b;
 }
 Var value = add (16, 10)
 return (
 <div classname = "App">
 <h1 > { value } </h1>
  </div>
  );
 }

export default App
```

Output :

This is Example of bundled React App
The Sum is : 26

Experiment-16

Aim:- Implement Simple ~~Hoj~~ Node.js app

Theory:- Node.js is an open source and cross-platform runtime environment for executing javascript code outside a browser. You need to remember that nodeJS is not a framework and its not a programming Language. We often use Nodejs for building back-end services like APIS like WebApp or Mobile App

Code:

```
console.log("This is the first Statement");
console.log("This is fourth Statement");
setTimeout(function(){
    console.log("This is second Statement");
},1000);
console.log("This is the third Statement");
```

Output:

Console:

This is first Statement
This is fourth Statement
This is third Statement
This is second Statement

Experiment - 17

Aim: Demonstrate REPL in Nodejs.

Theory: Read-Eval-Print-Loop (REPL) is an easy-to-use command line tool, used for processing Nodejs expressions. It captures the user's JavaScript code inputs, interprets and evaluates the result of this code. It displays the result to the screen and repeats the process until the user quits the shell.

Code and Output:

```
> 10+12
22
> "Vishal"
Vishal
> a=Vishal
Vishal
> a+17
> Vishal17
> 12|6
2
> 12x3
36
```

Experiment-18

Aim: Implement Streams and Buffers in Nodejs

Theory:- Streams are one of the fundamental concepts of Nodejs. Streams are a type of data-handling methods and are used to read or write input into output sequentially ~~Stream~~ Streams are used to handle/reading/writing files or exchanging information in an efficient way.
Buffer class is a global class that can be accessed in an application without importing the buffer module.

Code:

```
var fs = require("fs")
var data = " ";
var readStream = fs.createReadStream("exp1.html")
readStream.on("data", function(chunk){
 data += chunk;
});
readStream("end", function(){
 console.log(),
})
```

Output:

```
<!DOCTYPE>
<html>
<head>
<title>Document</title>
</head>
<body>
<h2>Hiii</h2>
<p>I'm Vishal</p>
<ul>hobbies
<li>gaming</li>
<li>drawing</li>
<ul>
</body>
</html>
```

Experiment - 19

Aim: Implement Nodejs web module

Theory: A web server is a software application which handles HTTP request sent by the HTTP client, like web browser, and returns web pages in response to the clients. Web servers usually deliver html documents along with images, style sheets and scripts

Code:

```
var http = require("http");
http.createServer(function(request, response)
{
  response.writeHead(200, {'content Type': 'text/son'});
  response.end('Node web module');
  }).listen(8081);
console.log("server running at http://121.0.0.1:8081");
```

Output:

128.0.1:8081

Nodejs web module

Experiment - 20

Aim: Implement Nodejs Net module

Theory: Nodejs provides the ability to perform Socket programming. Nodejs net module contains functions for creating both servers and clients.

Net Properties:

connect() Creates a new connection to the server, and returns a new Socket.

createConnection() Creates a new connection to the server, and returns a new Socket.

createServer() creates a new server is IP checks if the specified value is an IP address

Code:

```
//Server.js
var net = require("net");
var server = net.createServer(function(connection)
{
  console.log('client connected');
  connection.on('end', function(){
    console.log('client disconnected');
  });
connection.write("Hello world");
connection.pipe(connection);
});
server.listen(8081, function(){
```

```
  console.log ('Server is listening');
});

//client.js
var net = require ("net");
var client = net.connect ({port: 8081}, function (){
 console.log ('connected to server');
});
Client.on ('data', function(data){
 console.log (data.toString());
 client.end ();
});
Client.on ('end', function (){
  console.log ('disconnected from server');
});
```

Output:

Server.js	Client.js
Server is listening	Connected to server
Client connected	Hello world!
Client disconnected	disconnected from server

Experiment - 21

Aim: Access MongoDB database with Nodejs

Theory: The MongoDB Nodejs Driver allows you to easily interact with MongoDB databases from within Nodejs application. If you don't have the MongoDB Nodejs Driver installed, you can install it with npm install mongodb mongodb.connect() method. This is an asynchronous method of the Mongodb Module.

Code:

```
//Database.js
var MongoClient = require("mongodb").MongoClient;
var url = "mongodb://localhost:27017
MongoClient.connect(url, function(err,db){
  if (err) throw err;
  var dbo = db.db("VishalDB");
  var myobj = { name: "Vishal", rollno: 13};
  dbo.collection("Student").insertOne(myobj,
     function(err, res){
  if (err) throw err;
  console.log("1 document inserted");
  db.cols
  db.close();
  });
});
```

Output:

Terminal:

Database created!

1 document ~~ad~~ inserted

Database

Students → _id: ObjectId("....")
name: "Vishal"
rollno: 13

Experiment - 22

Aim: Implement Express router

Theory: The express Router() function is used to create a new router object. This function is used when you want to create a new router object in your program to handle requests

Code:

```
var express = require("express");
var app = express();
var router = express.Router();
var PORT = 3000;

router.use(function(req, res, next){
  console.log("Middleware called");
  next();
})
router.use(function(req, res, next){
res.send("Greeting from India");
})
app.use('/user', router);
app.listen(PORT, function(err){
  if (err) throw err;
  console.log("Server is listening on PORT", PORT);
});
```

Output:

Terminal:
Middleware called
Middleware called
Middleware called
Middleware called

Document:
Greeting from India

Experiment - 23

Aim: Create Rest API using Node js / Express

Theory: JavaScript Backend can be developed using Nodejs, Express. This backend can do Query operations on the PostgreSQL database and provide the status or data on the REST API.

Code:-

```
Var express = require('express');
Var app = express();

app.get('/hello', function(req, res){
  res.send("Hello world!");
});

app.post('/hello'; function(req, res){
 res.send("you just called a post method at
                '/hello'!");
  });

app.put('/hello,function(req, res){
res.send("you just called the put method at
                'hello',);
  });

  app.listen(3000);
```

Output:

Terminal:
Hello world!
You just called the POST method at '/hello'
You just called the PUT method at '/hello'

Document:
Hello world!

Experiment - 24

Aim: Implement generators in Express.

Theory: Socket.IO is a library for real time communication between the server and the client. In Socket.IO, the headers are shared only once and it also works on the top of the TCP layer. Web Sockets are the base of the Socket.IO library. It is easier to implement the Socket IO in express applications that are not formed with express-generator. Here the server or client emits an event and it is being caught by another one

Code and Output:

```
>npm install -g express-generator
> express --view=pug myapp
  ==
>cd myapp
> npm install
>SET DEBUG=myapp:* & npm start
> npm install
> myapp
> node ./bin/www
```

GET / 200 313.814 ms - 170

~~Exp~~

Document:

Express

Welcome to Express

Experiment - 25

Aim: Creating Session using Express

Theory: Session Management can be done in nodejs by using the express-session module. It helps in saving the data in the key-value form. In this module, the session data is not saved in the cookie itself, just the session ID.

Installation:
(i) npm install express-session
(ii) After that, you can create a folder and add a file for example index.js To run this file you need to run command.
node index.js

Code:
```
var express = require ("express");
var cookieparser= require ('cookie-parser');
var session = require ('express-session');

var app = express ();
app.use (cookieParser());
app.use (session({awesome: "Hey ! youare awesome"
}));
```

```
app.get('/', function (req, res){
if (req.session.page_views){
  req.session.page views++;
  res.send("You visited this page"+req.session.page
      +"times");
}
else{
req.session.page_view = 1;
res.send("Welcome to this page for the
          first time!")
  }
}
app.listen(3000);
```

Output : localhost:3000

Document :

Welcome to this page for the first time!

You have visited this page 5 times