Theory Assignment : 1

Name : Meet C. Patel

Roll No. : 051

Sem : 7th

Subject : Application Development using Full Stack (701)

1. Node.js : Introduction, features, execution, architecture.

→ Intro :-

- Node.js is an open-source, cross-platform, runtime env. built on Chrome's V8 JS engine. It allows developers to execute JS code outside the browser, making it possible to build server-side applications. Node.js enables developers to create scalable, efficient & real time application, and it has gained immense popularity in the web development community due to its non-blocking, event-driven architecture.

→ Features of Node.js:

1) Asynchronous and Non-Blocking :

- Node.js uses an event-driven, non blocking I/O model, allowing it to handle multiple concurrent connections efficiently without getting blocked by time-consuming operations.

2) Single-Threaded, Event-Loop Architecture :-

- Node.js operates on a single threaded event loop which enables it to handle asynchronous operation efficiently.

3) Fast Execution :

- Node.js is built on the V8 engine, which compiles JS code into highly efficient machine code, leading to faster execution. Compared to traditional interpreted language

4) NPM (Node Package Manager)
- NPM is a powerful package manage that comes bundled with node.js.

5) Cross platform:-
- Node.js is compatible with various operating systems, including windows, macOS and Linux, making it a versatile choice for building applications that can run on different platforms.

6) Scalability:-
- Due to its non-blocking nature, Node.js is highly scalable and can handle large number of concurrent connection with relatively low resource consumption.

→ Execution Arch:-

1) Event Loop:
- The event loop is the heart of Node.js responsible for handling all asynchronous operations.

2) Event Queue:-
- the event queue holds all the callbacks waiting to be executed. When an asynchronous operation completes, its callback is placed in the event queue.

3 Callback Func:-
- Callback Func are used to handle the result of asynchronous operations. When a task is completed, its corrosponding callbak is invoked by the event loop.

4) V8 engine:-
- Node.js uses the V8 JS engine to execute JS code. V8 compiles JS code for faster execution.

5) Libuv:-
- Libuv is a Library that provides the event loop and handles I/O operations, allowing Node.js to be cross-platform.

Q-2. Note on the modules with example.

→ In Node.JS, modules are used to encapsulate reusable pieces of code, making it easier to organize and maintain large applications.

- A module in Node JS is a self-contained piece of code that organized, promoting code reusablety and reducing namspace collis

→ Creating a Module.

```
- //rectangle.js
  const calculateArea = (width, height) => {
        return width * height;
  }; module.exports = calculateArea;
```

- Here we have created a module th calculates the area of rectangle.

→ Using a Module:-

- /app.js:-

```
cost calculateArea = require('./rectangle');
const width = 5;
const height = 10;

const area = calculateArea(width, height);

Console.log(" The area is ${area} ");
```

- We created a module 'rectangle.js' that exports the 'calculateArea' function. This func. takes 2 parameters. 'width' and 'height' and returns the calculated area. We exports the func using 'module.exports'. So that it can be used in other files.

- In 'app.js' file, we import the 'calculat Area' func. from 'rectangle.js module using 'require' function. the require func. is a built-in Node.js function used to load external model.

Q-3. Note on package with example.

→ In Node.js a package refers to as co
of modules, library and othe resources bundled
together and published on the npm registe

- Downloding a package is very easy.

- Here, I want to download a package called "upper-case". So in command Line Interface and download NPM.

C:\Users\My fc > npm install upper-case.

- NPM creates a folder named "node-module
where the packages will be placed. All the
package you install in the future will
placed in folder.

- New folder structure will be like this:

C:\Users\My PC\node-modules\upper-case

→ Using package:

- Once the package installed, it is read
to use.

- Include the "upper-case" package
same way you include any other module

Var uc = require ("upper-case");

- Create Node Js file that will convert "Hello world" into upper-case letter.

= var http = require('http');
var upc = require ("upper-case");
http createServer (function (req, res)
res. writeHead (200, {'Content-type': 'text-ht

res. write (uc. upperCase ("Hello World!"));
res. end ();
}). listen (8000);

- The output will be

Hello World!

Q-4. Use of package.json and package-lock 2)

→ Both 'package.json' and 'package-lock' are essential files in Node.js project. They serve different purposes but work together to manage project dependen and ensure consistent package installation across different environments.

→ package.json :— 3)

- The 'package.json' file is the manife file for a Node.js project. It contains metadata about the project includi → its name, version, description, entry point, scripts, dependencies and more

1) Dependency Management :—

- The 'dependencies' and 'devDependencies' section in packages required for the project to reun and for project to run and for development purpose. When someone clone your project to new environment they can run 'npm install' and npm will read the 'pack json' file to download and install all the required dependencies.

2) Scripts :-

- You can define custom scripts in the section of 'package.json'. These scripts can be executed using 'npm run <script-name>'. Common scripts include "start" to run the application and "test" to execute unit test.

3) Meta data:

- The 'package.json' file contain essential info. about the proj, such as author's name, proj description, license, Repository URL & more.

→ package-lock.json :-

- The 'packag-lock.json' file is automatically generated by npm when dependencies are installed or updated.

1) Dependency Locking:

→ The package-lock.json' file locks the installed package versions, including all their transitive dependencies. this ensures that everyone working on the project uses the exact same version mistches between development & production environment.

Q-5 Nodejs package:—

→ In the Node.js ecosystem, packages are collection of modules, libraries and resources that developers can use to enhance applications.

- Nodejs packages provide functionality various purposes, ranging from web development and server-side tasks to command-line utilities and more.

→ Web frameworks:—
- package like expressjs, koa and Hap popular web frameworks that simpli the process of building web applicati an API's by providind routing, middl support and other features.

→ Utility Libraries:—
- package like lodash, Ramda, and Und is provide utilitiey func. that assists with tasks like data manipulation, Validation and functional programming.

→ Database libraries:—
- Packages like Mongose and sequelize provide easy-to-use abstractions for working with databases and ORM capability.

→ Authentication and security:-
- Packages like Passport.js and decrypt. offer solutions for authentication and password hashing to enhance application security.

→ Template Engines:-
- Pakages like EJS, Handlebars, and Pug enable developers to generate dynamic HTML content easily.

→ HTTP Clients:
- Packages like Axios and Request provide tools for making HTTP requests, allowing Node.js application to iteract with APIs and web services.

→ Testing Frameworks:
- Package like Mocha, Jest and Chai offer testing utilities and assertion for creating and running test suits.

→ CLI:- (Command-Line-Interface) Tools:
- Pachages like Commander and yargs enable developers to build interactive and user-friendly command-line tools.

P.T.O

→ Real time Communication :-
- Packages like Socket.IO facilities rea
time communication between clients an
servers using Web Sockets

→ File System Utilities :-
- Package like fs-extra and glob pr
additional functionality and ease of
for working with the file system.

→ Date & time manipulation.
- Packages like Moment.js and Dev.js
offers tools for parsing, formatting a
manipulating dates & times.

2-6 NPM introduction & command with its use.

→ npm is the defaultt package manager for Node.js and it is one of the largest software Registeries in the world. It allows developers to easily install, manage, and distribute Nodejs packages to be used in their projects, npm comes bundled with node.js, so when you install Node.js, npm is automaticaly installed on your system.

→ npm init:
- This command initializes an new Node.js project and creates a 'package.json' file. It prompts you to enter details about the proj, such as name, version, description, author and entry point.

→ npm install : <package-name>
- Install a specific package and adds it to the 'dependencies' section in package.json'.

→ npm uninstall <Package name>
- Removes a Package from the proj. and updates the 'package.json' file accordingly.

→ npm - update:-
- Updates all the packages listed in 'package.json' to their latest version based on the specified version ranges.

→ npm search <package-name>
- serches the npm registery for packages with the given name.

→ npm ls:-
- Lists all the installed packages in current project.

→ npm init -y:-
- Initializes a new project with def values, skipping the prompts. It creates a 'package.json' file with default settings.

→ npm publish:-
- publishes a package to the npm registary making it available for oth to use.

Q-7. Describe use and working of following Node.js packages. Important properties and methods and relevant programs.

→

1) Url :-

- The 'url' module provides utilities for URL resolution and parsing. It is used to work with URLs & extract info from them.

Eg. Parsing a URL

```
const {URL} = requires ('url');
Const urlstring = "https://www.demo.com:8000/path?
                                    query = hellow
const parsedURL = new URL (urlstring);

Console.log (parsedURL.host);
Console.log (parsedURL.pathname);
Console.log (parsedURL.searchParams.get ('query'));
```

2) process, pm2 (external package):

- 'process' object provides info & control over the Node.js process. It allows interacting with the current process and accessing environment variable.
- Getting Command-line Arguments.

```
Console.log (process.argv);
```

- 'pm2' is an external package used to manage Node.js processes. It provides tools for process monitoring, scaling and cluster management.

- # Install pm2 globally
  npm install -g pm2
  pm2 start app.js

3) Readline:
- The 'readline' module provides an interface for reading input streams line by line. It commonly used to interact with users in command line environment.
- Reading Users Input.

```
const readLine = require('readLine');
const rl = readline.createInterface({
    input : process.stdin;
    output : process.stdout;
});
rl.question('what's your name?', (name)
{
    console.log('Hello, ${name}!')
    rl.close();
});
```

4) 'fs':-
- The 'fs' module provides file system-related functionality, allowing reading, writing & manipulating files.

- Reading a file :-

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

fs.readFile('example.txt', 'utf8', (err, data) => {
  if (err) {
    console.error("Error reading file", err);
    return;
  }
  console.log('File content', data);
});
```

5) events:

- The 'events' module module provides an event-driven architecture for building applications that can emit and listen to events.

```
- const EventEmitter = require('events');
  class MyEmitter extends EventEmitter {}

  const myEmitter = new MyEmitter();
  myEmitter.on('greet', (name) => {
    console.log('Hello', ${name}!');
  });

  myEmitter.emit('greet', 'Meet');
```

6) Console :—

- The 'console' module provides a simple debugging console that can be used to log messages during development.

```
console.log ('this is a log message')
Console.error ('this is an error messag
Console.warn ('this is a warning mess
```

7) buffer:—

- The 'buffer' module provides a way to hu binary data. It is used to work with raw binary data in node.js applicatio

```
Var buf = Buffer.from ('abc');
Console.log (buf);
```

Ans : < Buffer 61 62 63 >

8) 'querystring' module provedes utilities fo working with query strings in URLs

```
-const querystring = require ("querystring")

Const params = querystring.parse ('name: Moet
                                   age = 20');

console.log (params)
```

- Output :—

{ name = 'moet', age : '20' }

a) http:-
- The 'http' module provides classes to create HTTP requests and make HTTP requests.

```
const http = require('http');
const server = http.createServer((req, res) => {
    res.writeHead(200, { 'Content-Type': 'te
    res.end("Hello World!");
});
server.listen(3000, () => {
    console.log('server is running on
});
```

b) V8:
- The 'V8' module exposes APIs related to the V8 js engine, providing access performance & memory-related data.

c) Os:-
- The 'os' module provides os related fu such as info about the host os.
- const os = require('os');

```
console.log('os Platform:', os.platform()
console.log("CPU Architecture:", os.arch()
```

9) http :-
- The 'http' module provides a set of functions and classes to create HTTP servers & make HTTP requests.

```
const http = require('http');
Const server = http.createServer((req, res) =>
{
    res.writeHead(200, { 'Content-Type': 'text/plain' })
    res.end("Hello World!");
});
server.listen(3000, () => {
    console.log('server is running on port 3000
});
```

10) V8 :
- The 'V8' module exposes APIs related to the V8 js engine, providing access to performance & memory-related data.

11) Os :-
- The 'os module provides os related functions such as info about the host os.

```
- const os = require('os');

Cosole.log('os Platform:', os.platform());
console.log('CPU Architecture:', os.arch());
```

12) zlib :-

- The 'zlib' module provides compression and decompression functionalities using gzip & deflate.

- Comprasing & Decompressing :-

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

const data = 'This is same data to comp

zlib.gzip(data, (err, compressedData) => {
  if (err) {
    console.error('compression error :', err);
    return;
  }

  console.log("compressed data:', compressedda

  zlib.unzip(compressedData, (err, decompressedDat
  {
    if (err) {
      console.error('Decompression error', err)
      return;
    }
    console.log('Decompressed data:',
                decompressedData.toString(
  });
});
```

Theory Assignment : 1

Name : Meet C. Patel

Roll No. : 051

Sem : 7th

Subject : Application Development using Full Stack (701)