Page No.
Date

Theory Assignment - 1

Name: Yug P. Parmar

Roll No.: 42

Sem: 7th

Subject: Application Development using Full Stack (702)

Theory Assignment - 1

Q-1. Nodejs : Introduction, features, execution architecture.

→ Introduction:-

- Node.js is an open-source, cross-platform, runtime environment built on Chrome's V8 Javascript engine. It allows developers to to execute JavaScript code outside the browser, making it possible to build server-side applications. Node.js enables developers to create scalable, efficient, and real-time application, and it has gained immense popularity in the ~~web~~ 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 operations efficiently.

3) Fast Execution:-
- Node.js is built on the V8 engine, which compiles JavaScript code into highly efficient machine code, leading to faster execution compared to traditional interpreted languages.

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

5) ~~Large Comma~~ 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 platform.

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

→ Execution Architecture:-

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 Functions:-
- Callback functions are used to handle the results of asynchronous operations. When a task is completed, its corresponding callback is invoked by the event loop.

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

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 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 ~~res~~ reusability, and reducing namespace collisions.

→ Creating a Module.

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

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

→ Using a Module:-

```
// app.js

const 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 function takes two parameters, 'width' and 'height', and returns the calculated area. We export the function using 'module.exports' so that it can be used in other files.
- In 'app.js' file, we import the 'calculateArea' function from 'rectangle.js' module using 'require' function.
- The 'require' function is a built-in Nodejs functioned used to load external modules.

Q-3 Note on package with example.

→ In Node.js, a package refers to a collection of modules, libraries and other resources bundled together and published on the npm registery.

- Downloading a package is very easy.

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

C:\Users\MyPC > npm install upper-case

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

- Now folder structure will be like this:

C:\Users\MyPC\node-modules\upper-case

→ Using a Package :-

- Once the package is installed, it is ready to use.
- Include the "upper-case" package the same way you include any other module:

```
var uc = require ('upper-case');
```

- Create a Node.js file that will convert "Hello World!" into upper-case letter:

```
var http = require ('http');
var uc = require ('upper-case');
http.createServer (function (req, res)
{
    res.writeHead (200, {'Content-Type': 'text-html'});
    res.write (uc.upperCase ("Hello world!"));
    res.end ();
}).le listen (8080);
```

- The output will be :-

HELLO WORLD!

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

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

→ package.json:-

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

1) Dependency Management:-

- The 'dependencies' and 'devDependencies' sections in 'package.json' list the packages required for the project to run and for development purposes, respectively. When someone clones your project or deploys it to a new environment,

they can run 'npm install', and npm will read the 'package.json' file to download and install all the required dependencies.

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

3) Metadata:-
- The 'package.json' file contains essential information about the project, such as author's name, project description, ~~licence~~ license, repository URL, and more.

→ package-lock.json:-

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

1) Dependency Locking :-
The 'packag-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 versions of packages, preventing their version mismatches between development and production environments.

2) Faster Installation :-
The 'packag-lock.json' file allows npm to perform faster and more efficient installations by knowing the exact versions of packages needed, without having to search for compatible versions in the registery.

Q-5 Nodejs packages.

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

- Node.js packages provide functionalities for various purposes, ranging from web development and server-side tasks to command-line utilities and more.

→ Web Frameworks:-

- Packages like Express.js, Koa and Hapi are popular web frameworks that simplify the process of building web applications and APIs by providing routing, middleware support, and other features.

→ Utility Libraries:-

- Packages like Lodash, Ramda, and Underscore.js provide utility functions that assist with tasks like data manipulation, validation, and functional programming.

→ Database Libraries:-
- Packages like Mongoose & and Sequelize provide easy-to-use abstractions for working with databases and ORM (Object-Relation Mapping) capabilities.

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

→ Template Engines:-
- Packages 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 applications to interact with APIs and web services.

→ Testing Frameworks:
- Packages like Mocha, Jest, and Chai offer testing utilities and assertions for for creating and running test suites.

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

→ Real-time Communication:
- Packages like Socket.IO facilitate real-time communication between clients and servers using WebSockets.

→ File System Utilities:-
- Packages like fs-extra and glob provide additional functionalities and ease of use for working with the file system.

→ Date and Time Manipulation:
- Packages like Moment.js and Day.js offer tools for parsing, formatting, and manipulating dates and times.

→ Logging:-
- Packages like Winston and Bunyan provide flexible logging solutions for Node.js applications.

Q-6 NPM introduction and commands with its use.

→ npm is the default 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 Node.js packages to be used in their projects. npm comes bundled with Node.js, so when you install Node.js, npm is automatically installed on your system.

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

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

→ npm uninstall <packagename>:-
- Removes a package from the project and updates the 'package.json' file accordingly

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

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

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

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

→ npm publish:-
- Publishes a package to the npm registry, making it available for others to use.

Q-7 Describe use and working of following Nodejs 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 and extract information from them.

Eg: Parsing a URL:-

```
const {URL} = requires ('url');

const urlString = 'https://www.demo.com:8080/path?query=hello';
const parsedUrl = new URL(urlString);

Console.log (parsedURrl.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 is commonly used to interact~~ive~~ with users in the command-line environment.
- Reading User 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, and 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', 'Yug');
```

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 message.');
console.warn('This is a warning message');
```

7) buffer:-

- The 'buffer' module provides a way to handle binary data. It is used to work with raw binary data in Node.js applications.

```
var buf = Buffer.from('abc');
console.log(buf);
```

- Ans: <Buffer 61 62 63>

8) querystring:-

- The 'querystring' module provides utilities for working with query strings in URLs.

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

const params = querystring.parse('name=Yug&age=20');
console.log(params)
```

- Output:-

```
{name: 'Yug', age: '20'}
```

9) http:-

- The 'http' module provides a set of functions and classes to create HTTP servers and 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 javascript engine, providing access to performance and memory-related data.

```
const v8 = require('v8');
console.log(v8.getHeapStatistics());
```

11) os:-

- The 'os' module provides operating system related functionality, such as information about the host operating system.

```
const os = require('os');
console.log('OS platform:', os.platform());
console.log('CPU Architecture:', os.arch());
```

#12) zlib :-

- The 'zlib' module provides compression and decompression functionalities using gzip and deflate.
- Compressing and Decompressing :-

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

const data = 'This is some data to compress.';
zlib.gzip(data, (err, compressedData) => {
  if(err) {
    console.error('Compression error:', err);
    return;
  }
  console.log('Compressed data:', compressedData);

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