Write-up	Correctnessof program	Documentation of program	Viva	Timely Completion	Total	Dated sign of Subject Teacher
2	2	2	2	2	10	

Assignment No 3
(A)

Aim: Create a Node.JS Application which serves a static website.

Theory:

Introduction to Node.js: -

For a long time, the browser was the only place JavaScript code could be executed. Web developers had to use a different programming language on the front-end than the back end. It also meant that, even as JavaScript evolved into a more robust and powerful language, it remained a front-end only language. Though multiple attempts to create off-browser JavaScript environments have been attempted, Node.js, invented by Ryan Dahl in 2009, found unprecedented popularity and is currently being used by numerous top-tier companies including Netflix, Uber, PayPal, and eBay. Node.js is a JavaScript runtime, or an environment that allows us to execute JavaScript code outside of the browser. A "runtime" converts code written in a high-level, human-readable, programming language and compiles it down to code the computer can execute.

- Node.js is an open-source and cross-platform JavaScript runtime environment. It is a popular tool for almost any kind of project!
- Node.js runs the V8 JavaScript engine, the core of Google Chrome, outside of the browser. This allows Node.js to be very performant.
- A Node.js app runs in a single process, without creating a new thread for every request. Node.js provides a set of asynchronous I/O primitives in its standard library that prevent JavaScript code from blocking and generally, libraries in

Node.js are written using non-blocking paradigms, making blocking behavior the exception rather than the norm.

Why Node.js?

A common task for a web server can be to open a file on the server and return the content to the client.

Here is how PHP or ASP handles a file request:

- Sends the task to the computer's file system.
- Waits while the file system opens and reads the file.
- Returns the content to the client.
- Ready to handle the next request.

Here is how Node.js handles a file request:

- Sends the task to the computer's file system.
- Ready to handle the next request.
- When the file system has opened and read the file, the server returns the content to the client.
- Node.js eliminates the waiting, and simply continues with the next request.
- Node.js runs single-threaded, non-blocking, asynchronous programming, which is very memory efficient.

How to Serve Static Content using Node.js?

Accessing static files are very useful when you want to put your static content accessible to the server for usage. To serve static files such as images, CSS files, and JavaScript files, etc. we use the built-in middleware in node.js i.e., express.static.

Step 1 — Setting up Express

To begin, run the following in your terminal:

Create a new directory for your project named "assignment3a":

$ mkdir assignment3a

Change into your new directory:

$ cd assignment3a

Initialize a new Node project with defaults. This will set a package.json file to access your dependencies:

$ npm init -y

Create your entry file, index.js. This is where you will store your Express server:

$ touch index.js

Install Express as a dependency:

$ npm install express --save

Within your package.json, update your start script to include node and your index.js file.

package.json

```
{
  "name": "assignment3a",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
                "test": "echo \"Error: no test specified\" && exit 1"
            },
  "author": "",
  "license": "ISC",
  "dependencies": {
                      "express": "^4.17.2"
                  }
}
```

This will allow you to use the npm start command in your terminal to launch your Express server.

Step 2 — Structuring Your Files

To store your files on the client-side, create a public directory and include an index.html file. Your file structure will look like this:

assignment3a
|- index.js
|- public
|- index.html

Now that your files are set up let's begin your Express server.

Step 3 — Creating Your Express Server

In your index.js file, require in an Express instance and implement a GET request:

index.js

```
const express = require('express');
const app = express();
const port = 4000;
app.listen(port, () => {console.log(`app started on port ${port}`);
```

- Now let's tell Express to handle your static files. Express provides a built-in method to serve your static files: When you call app.use(), you're telling Express to use a piece of middleware. *Middleware* is a function that Express passes requests through before sending them to your routing functions. express.static() finds and returns the static files requested. The argument you pass into express.static() is the name of the directory you want Express to serve files. Here, the public directory.
- In index.js, serve your static files below your PORT variable. Pass in your public directory as the argument:

```
app.use(express.static("public"));
```

With your Express server set, let's focus on the client-side.

Step 4 — Building Your Web Page

Navigate to your index.html file in the public directory. Populate the file with body and image elements.

Step 5 — Running Your Project

In your terminal, launch your Express project:

$ npm start

Server listening on port: 4000

Open your web browser and navigate to http://localhost:4000. You will see your project:

Conclusion:

Express offers a built-in middleware to serve your static files and modularizes content within a client-side directory in one line of code. Thus, we have created a Node.JS Application which serves a static website.