A-MEAN Practical Write Up 1-10

Practical No 1

Aim : Installation and initialization of Node.js

Node.js is a JavaScript runtime environment used for developing server-side applications. In this practical, Node.js was downloaded from the official website and installed using the setup wizard. After installation, its version was checked through the terminal to confirm successful setup. The initialization process was then carried out by creating a new project folder and using basic commands to generate the necessary project files. This practical helped in understanding the installation process of Node.js and how to begin working with it for backend development.

Practical No. 2

Aim: Write a program to demonstrate creation and understanding of package.json and usage of npm.

In this practical, a new Node.js project was initialized to automatically generate the package.json file, which stores essential project details and dependencies. The usage of npm was explored to install and manage packages required in a Node.js application. Basic npm commands were used to understand how dependencies are added and maintained. This practical provided a clear understanding of how package.json and npm help in organizing and managing Node.js projects efficiently.

Practical No. 3

Aim: Write a program to demonstrate creating a module and using module export or require.

In this practical, the concept of modular programming in Node.js was demonstrated by creating a separate module and importing it into another file. A custom module was created to perform an addition operation, and it was exported so it could be used externally. The main file then imported this module using the require function and utilized the exported function within a basic server setup. This practical showed how modules help in organizing code, improving reusability, and maintaining a clean project structure in Node.js applications.

Practical No. 4

Aim: Write a program to demonstrate the Node.js event loop.

In this practical, different phases of the Node.js event loop were demonstrated using various asynchronous functions such as *setTimeout*, *setInterval*, *setImmediate*, *process.nextTick*, and the *async–await* mechanism. Each of these functions was observed to understand when and how they are executed within the event loop cycle. The practical showed the order in which synchronous and asynchronous tasks run, highlighting how Node.js manages callbacks, timers, microtasks, and promises. Through these examples, the working of the event loop became clear, along with its importance in handling non-blocking operations efficiently in Node.js applications.

Practical No. 5

Aim: Write a program to demonstrate creating a basic HTTP server.

In this practical, a simple HTTP server was created using Node.js to understand how server-side communication works. The built-in *http* module was used to create a server that listens for incoming requests and sends a basic response. When the server receives a request, it returns a message and then ends the response. The server was configured to run on a specific port, showing how Node.js handles network connections efficiently. This practical provided a clear understanding of how to set up and run a basic HTTP server in Node.js.

Practical No. 6

Aim: Write a program to demonstrate handling routes and requests in a Node.js server.

In this practical, different routes were created and handled within a Node.js server to understand how client requests are processed based on the URL. Using the built-in *http* module, the server identified the requested path and responded with the appropriate message for each route such as home, about, add, and contact. A custom module was also used to perform a simple calculation on one of the routes. If an undefined route was accessed, a “Page Not Found” message was returned. This practical helped in understanding how routing works and how servers handle multiple requests in a structured manner.

Practical No. 7

Aim: Write a program to demonstrate introduction and creation of an Express.js application.

In this practical, the basics of Express.js were introduced by creating a simple server application. Express.js, a lightweight and flexible Node.js framework, was used to simplify the process of handling requests and responses. A new Express app was set up, and a basic route was created to send a response when the home page was accessed. The server was then started on a specific port, demonstrating how easily Express.js can be used to build and manage web applications. This practical provided a clear understanding of how to initialize and run a basic Express.js application.

Practical No. 8

Aim: Write a program to demonstrate the use of middleware in Express.js.

In this practical, the concept of middleware in Express.js was demonstrated. Middleware functions are used to process requests before they reach the core application logic. An Express application was created, and middleware was added to handle tasks such as JSON parsing and logging of incoming requests. This practical helped in understanding how middleware can manage operations like logging, authentication, and data processing, improving the modularity and structure of Express.js applications.

Practical No. 9

Aim: Write a program to demonstrate routing in Express.js.

In this practical, routing in Express.js was demonstrated by creating multiple routes in a server application. The Express framework was used to handle different URL paths and send appropriate responses for each route. This practical helped in understanding how routing allows an application to respond differently based on client requests, making it possible to organize and manage multiple endpoints efficiently in an Express.js application.

Practical No. 10

Aim: Write a program to demonstrate RESTful API in Express.js (GET, POST, PUT, DELETE).

In this practical, a RESTful API was created using Express.js to handle basic CRUD operations on user data. Different HTTP methods were demonstrated as follows:

- **GET:** Used to retrieve existing data from the server.
- **POST:** Used to add new data to the server.
- **PUT:** Used to update existing data on the server.
- **DELETE:** Used to remove data from the server.

Express.js and JSON parsing middleware were used to manage request data efficiently. This practical helped in understanding how RESTful APIs operate, how different HTTP methods are used for specific operations, and how Express.js simplifies building and managing API endpoints.