**Assignment-5**

1. **What is Nodejs? Explain it's 5 uses?**

* **Node.js** is a runtime environment that allows you to run JavaScript code on the server side. It uses the V8 JavaScript engine, developed by Google, which executes JavaScript code outside of a web browser. Node.js provides an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for building scalable and real-time applications.
* **Here are five common uses of Node.js:**
* **Web Development:** Node.js is widely used for building web applications and APIs. Its event-driven architecture and non-blocking I/O make it suitable for handling numerous concurrent connections. Popular frameworks like Express.js are often used in conjunction with Node.js to simplify web development tasks.
* **Real-time Applications:** Node.js is well-suited for building real-time applications like chat applications, online gaming platforms, collaborative tools, and live-tracking applications. Its event-driven architecture allows for handling multiple client requests simultaneously, making it ideal for applications that require instant data updates.
* **Microservices:** Node.js is commonly used in microservices architectures due to its lightweight nature and scalability. Developers can build independent, smaller services that communicate with each other, allowing for easier maintenance and scaling of complex applications.
* **Command Line Tools:** Node.js can be utilized to create command-line tools and scripts, making it a versatile choice for automating repetitive tasks, system administration, and development workflows. Packages like Commander.js and yargs simplify the process of building command-line interfaces.
* **IoT (Internet of Things):** Node.js can be used in IoT applications due to its lightweight footprint and ability to handle asynchronous operations effectively. It can manage devices, collect data from sensors, and interact with hardware, making it suitable for developing IoT solutions.

1. **Why do we use Nodejs? How it handle request?**

* **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.

1. **What Nodejs can perform? Write about command line interface?**

* **Node.js can Perform:**
* Node.js can generate dynamic page content
* Node.js can create, open, read, write, delete, and close files on the server
* Node.js can collect form data
* Node.js can add, delete, modify data in your database
* **Node.js File means:**
* Node.js files contain tasks that will be executed on certain events
* A typical event is someone trying to access a port on the server
* Node.js files must be initiated on the server before having any effect
* Node.js files have extension ".js"
* **Command Line Interface:** Node.js files must be initiated in the "Command Line Interface" program of your computer. How to open the command line interface on your computer depends on the operating system. For Windows users, press the start button and look for "Command Prompt", or simply write "cmd" in the search field.

1. **Explain Nodejs process model? Explain with diagram?**

|  |
| --- |
|  |

* **Node.js Process Model:** In the traditional web server model, each request is handled by a dedicated thread from the thread pool. If no thread is available in the thread pool at any point of time then the request waits till the next available thread. Dedicated thread executes a particular request and does not return to thread pool until it completes the execution and returns a response.
* Node.js processes user requests differently when compared to a traditional web server model. Node.js runs in a single process and the application code runs in a single thread and thereby needs less resources than other platforms.
* All the user requests to your web application will be handled by a single thread and all the I/O work or long running job is performed asynchronously for a particular request.
* So, this single thread doesn't have to wait for the request to complete and is free to handle the next request. When asynchronous I/O work completes then it processes the request further and sends the response
* **The following figure illustrates asynchronous web server model using Node.js:**

|  |
| --- |
|  |

* Node.js process model increases the performance and scalability with a few caveats. Node.js is not fit for an application which performs CPU-intensive operations like image processing or other heavy computation work because it takes time to process a request and thereby blocks the single thread.
* Contrary to the traditional web server model, NodeJS uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. The NodeJS process model can be explained with three architectural features of NodeJS.

1. **What is Nodejs module? Explain any 2 module in details?**

* **Node.js Module:** Module in Node.js is a simple or complex functionality organized in single or multiple JavaScript files which can be reused throughout the Node.js application. Each module in Node.js has its own context, so it cannot interfere with other modules or pollute global scope. Also, each module can be placed in a separate .js file under a separate folder. Node.js implements [CommonJS modules standard](http://requirejs.org/docs/commonjs.html). CommonJS is a group of volunteers who define JavaScript standards for web server, desktop, and console application. Node.js includes three types of modules: **Core Modules, Local Modules, Third Party Modules.**
* **Node.js Modules:** Node.js is a light weight framework. The core modules include bare minimum functionalities of Node.js. These core modules are compiled into its binary distribution and load automatically when Node.js process starts. However, you need to import the core module first in order to use it in your application.

|  |  |
| --- | --- |
| **Module** | **Description** |
| [**http**](https://nodejs.org/api/http.html) | http module includes classes, methods and events to create Node.js http server. |
| [**url**](https://nodejs.org/api/url.html) | url module includes methods for URL resolution and parsing. |
| [**querystring**](https://nodejs.org/api/querystring.html) | querystring module includes methods to deal with query string. |
| [**path**](https://nodejs.org/api/path.html) | path module includes methods to deal with file paths. |
| [**fs**](https://nodejs.org/api/fs.html) | fs module includes classes, methods, and events to work with file I/O. |
| [**util**](https://nodejs.org/api/util.html) | util module includes utility functions useful for programmers. |

1. **What file system in Nodejs? Explain it's operation in detail?**

* **In Node.js,** the File System (FS) module provides a way to interact with the computer's file system. It allows you to perform various operations on files and directories, such as creating, reading, updating, deleting, and renaming. The FS module is built on asynchronous functions, which means that operations are performed in a non-blocking manner, allowing your code to continue executing while the operation is in progress.
* The "file system" generally refers to the module named fs (short for "file system"). This module provides an API for interacting with the file system in a Node.js application. It allows you to perform various operations on files, directories, and other file-related tasks.
* The “fs” module includes methods to perform both synchronous and asynchronous operations. Some of the commonly used operations include:
* **File Operations:**
* Reading files: fs.readFile(), fs.readFileSync()
* Writing files: fs.writeFile(), fs.writeFileSync()
* Deleting files: fs.unlink(), fs.unlinkSync()
* Renaming files: fs.rename(), fs.renameSync()
* Checking file existence: fs.existsSync()
* **Directory Operations:**
* Creating directories: fs.mkdir(), fs.mkdirSync()
* Reading directories: fs.readdir(), fs.readdirSync()
* Removing directories: fs.rmdir(), fs.rmdirSync()
* The file system operations provided by the fs module are essentially abstractions over underlying system calls provided by the operating system. When you perform a file system operation in Node.js using fs, it interacts with the operating system to execute these low-level system calls.
* Internally, Node.js leverages the system's native capabilities to perform file-related tasks. For instance, when you read from a file using fs.readFile(), Node.js internally makes system calls to the operating system's file I/O functions to retrieve the data from the specified file.

1. **Describe database in Nodejs & Explain?**
2. **Database connectivity:** Database connectivity in Node.js involves establishing a connection between a Node.js application and a database server. This connection allows the application to perform various operations on the database, such as reading, writing, updating, and deleting data. There are several popular database drivers available for Node.js, including:

* **MySQL:** A relational database management system (RDBMS) that stores data in tables with rows and columns.
* **MongoDB:** A NoSQL database that stores data in JSON-like documents.
* **PostgreSQL:** An open-source RDBMS known for its reliability and performance.
* **Redis:** An in-memory data store that provides high-speed caching and data access.

1. **Connection string:** A connection string is a string of text that contains the necessary information to connect to a database server. It typically includes the following elements:

* **Hostname:** The IP address or domain name of the database server.
* **Port:** The port number that the database server is listening on.
* **Database name:** The name of the database to connect to.
* **Username:** The username to connect to the database with.
* **Password:** The password for the database username.
* For example, a connection string for a MySQL database might look like this:

|  |
| --- |
| **mysql://username:password@hostname:port/database\_name** |

1. **Database operation:** Once a connection to the database is established, Node.js applications can perform various operations on the data. These operations can be broadly categorized into the following types:

* **Creating:** Inserting new data into the database.
* **Reading:** Retrieving existing data from the database.
* **Updating:** Modifying existing data in the database.
* **Deleting:** Removing data from the database.
* These operations are typically performed using the database driver's API, which provides methods for each type of operation. For example, to create a new user in a MySQL database, you might use the following code:

|  |
| --- |
| **const mysql = require('mysql');**  const connection = mysql.createConnection({  host: 'localhost',  port: 3306,  database: 'mydatabase',  user: 'root',  password: 'password'  });  connection.connect(err => {  if (err) throw err;  const sql = 'INSERT INTO users (username, email) VALUES (?, ?)';  const values = ['johndoe', 'johndoe@example.com'];  connection.query(sql, values, (err, result) => {  if (err) throw err;  console.log('User created successfully');  });  connection.end();  }); |

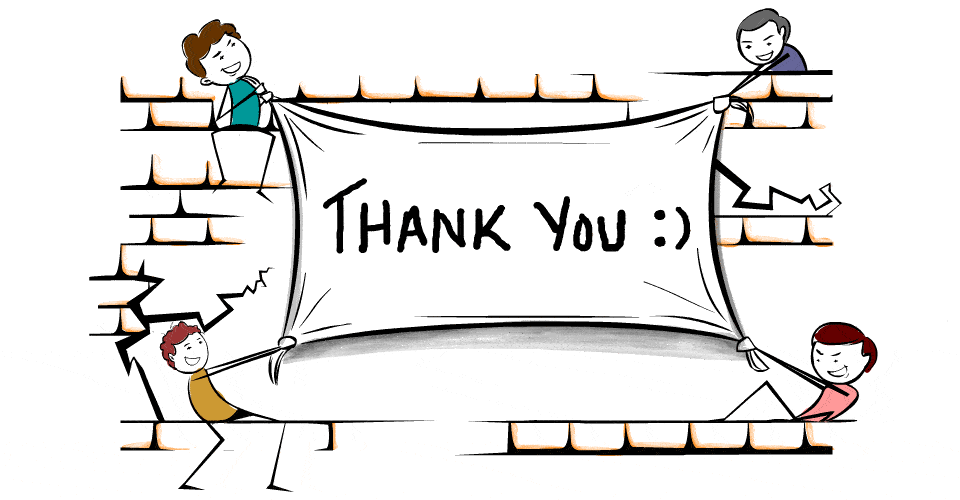
This code connects to a MySQL database, creates a new user record, and then closes the connection.

|  |
| --- |
|  |

1. **Write Short note on MERN?**

* **MERN** Stack is a JavaScript Stack that is used for easier and faster deployment of full-stack web applications. MERN Stack comprises of 4 technologies namely: MongoDB, Express, React and Node.js. It is designed to make the development process smoother and easier.
* **MongoDB:** Non Relational Database
* **Express:** Node.js web server
* **React:** JavaScript Frontend Framework
* **Node:** JavaScript Web Server
* **How MERN stack works:** When working with MERN stack, developers create implement View layer using React and Express and Node are used to implement application layer of website then MongoDB is used to implement database layer.
* **Roadmap to become a MERN Stack Developer:**
* **Step 1:** Learn basics of HTML, CSS and JavaScript
* **Step 2:** Learn React which is a frontend library for building User Interfaces
* **Step 3:** Learn Node.js which is JavaScript runtime environment
* **Step 4:** Learn Express.js, a framework built upon Node.js to simplify the process of creating web application and API building.
* **Step 5:** Learn MongoDB, a NoSQL database to store and retrieve data from database.

**(PPT Give you more understanding than PDF)** The material for the PDF has been compiled from various sources such as books, tutorials (offline and online), lecture notes, several resources available on Internet. The information contained in this PDF is for general information and education purpose only. While we endeavor to keep the information up to date and correct, **we make no representation of any kind about the completeness and accuracy of the material.** The information shared through this PDF material should be used for educational purpose only.

****