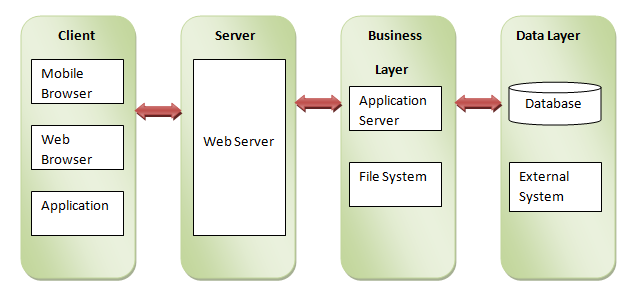
**1. What is Nodejs and Explain the architecture and internal working of NodeJS with an example?**

Node.js is Server-side scripting which is used to build scalable programs. It is a web application framework built on Google Chrome's JavaScript Engine. It runs within the Node.js runtime on Mac OS, Windows, and Linux with no changes. This runtime facilitates you to execute a JavaScript code on any machine outside a browser.

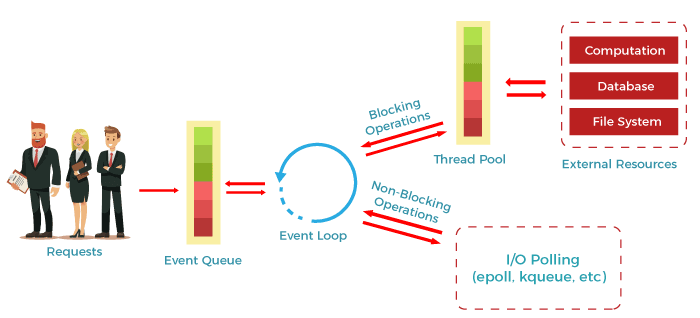
Architecture of NodeJS

A web application distinguishes into 4 layers:

* **Client Layer:** The Client layer contains web browsers, mobile browsers or applications which can make an HTTP request to the web server.
* **Server Layer:** The Server layer contains the Web server which can intercept the request made by clients and pass them the response.
* **Business Layer:** The business layer contains application server which is utilized by the web server to do required processing. This layer interacts with the data layer via database or some external programs.
* **Data Layer:** The Data layer contains databases or any source of data.



Working of NodeJS



* According to the above diagram, the clients send requests to the webserver to interact with the web application. These requests can be non-blocking or blocking and used for querying the data, deleting data, or updating the data.
* js receives the incoming requests and adds those to the Event Queue.
* After this step, the requests are passed one by one through the Event Loop. It checks if the requests are simple enough not to require any external resources.
* The event loop then processes the simple requests (non-blocking operations), such as I/O Polling, and returns the responses to the corresponding clients.
* A single thread from the Thread Pool is assigned to a single complex request. This thread is responsible for completing a particular blocking request by accessing external resources, such as computation, database, file system, etc.
* Once the task is completed, the response is sent to the Event Loop that sends that response back to the client.

**2. What is module in Nodejs? What are the core modules in Nodejs explain each, and**

**explain the process of installing, initializing of modules?**

In Node.js applications, modules are like JavaScript libraries and include a set of functions. To include a module in a Node.js application, we must use the require() function with the parentheses containing the module's name.

Node.js has several modules which are used to provide the basic functionality needed for a web application. Following is a list of some of them:

Core Modules Description

**HTTP**: The HTTP module includes classes, methods, and events to create a Node.js HTTP server.

**util**: The util module includes utility functions required in the application and is very useful for developers.

**url**: The url module is used to include the methods for URL parsing.

**fs**: The fs module includes events, classes, and methods to handle the file I/O operations.

**stream**: The stream module is used to include the methods to handle streaming data.

**query string**: The query string module is used to include the methods to work with a query string.

**zlib**: The zlib module is used to include the methods to compress or decompress the files used in an application.

Loading Core Modules

In order to use Node.js core or NPM modules, you first need to import it using require() function as shown below.

var module = require('module\_name');

As per above syntax, specify the module name in the require() function. The require() function will return an object, function, property or any other JavaScript type, depending on what the specified module returns.

The following example demonstrates how to use Node.js http module to create a web server.

Example: Load and Use Core http Module

 Copy

var http = require('http');

var server = http.createServer(function(req, res){

//write code here

});

server.listen(5000);

In the above example, require() function returns an object because http module returns its functionality as an object, you can then use its properties and methods using dot notation e.g. http.createServer().In this way, you can load and use Node.js core modules in your application.

**3. What is npm and why we use npm explain briefly?**

npm stands for Node Package Manager. Following are the two main functionalities of npm:

* Online repositories for node.js packages/modules which are searchable on search.nodejs.org
* Command line utility to install packages, do version management and dependency management of Node.js packages.

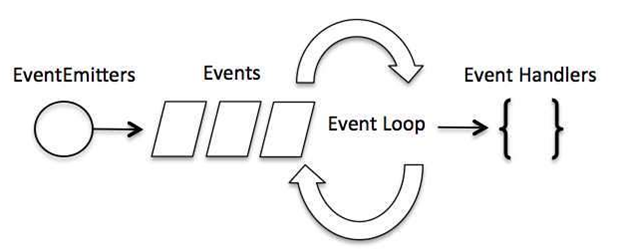
It helps with installing various packages and resolving their various dependencies. It greatly helps with your Node development. NPM helps you install the various modules you need for your web development and not just given you a whole bunch of features you might never need.

**4. What is callback function and Event driven programming?**

A callback function, also known as a higher-order function, is a function that is passed to another function (let's call this other function “otherFunction”) as a parameter, and the callback function is called (or executed) inside the otherFunction.

Event-driven programming is a programming paradigm in which the flow of program execution is determined by events - for example a user action such as a mouse click, key press, or a message from the operating system or another program. Virtually all object-oriented and visual languages support event-driven programming.

In Node.js, event-driven programming means as soon as Node starts its server, it initiates its variables, declares functions and then waits for an event to occur. It is one of the reasons why Node.js is pretty fast compared to other similar technologies.



**5. What is filesystem and why we use?**

Node implements File I/O using simple wrappers around standard POSIX functions. The Node File System (fs) module can be imported using the following syntax −

var fs = require("fs")

To handle file operations like creating, reading, deleting, etc., Node.js provides an inbuilt module called FS (File System). Node.js gives the functionality of file I/O by providing wrappers around the standard POSIX functions. All file system operations can have synchronous and asynchronous forms depending upon user requirements.

To use this File System module, use the require() method

Common use for the File System module:

* Read files
* Create files
* Update files
* Delete files
* Rename files

**6. Explain briefly about crypto and path modules?**

The Node.js crypto module provides cryptographic functions to help you secure your Node.js app. It includes a set of wrappers for OpenSSL’s hash, HMAC, cipher, decipher, sign, and verify functions. The crypto module is mostly useful as a tool for implementing cryptographic protocols such as TLS and https. For most users, the built-in tls module and https module should more than suffice. However, for the user that only wants to use small parts of what's needed for full-scale cryptography or is crazy/desperate enough to implement a protocol using OpenSSL and Node.js: Read on.

Node.js path module is used for handling and transforming file paths. This module can be imported using the following syntax.

var path = require("path")

Methods

Sr.No. Method & Description

1

path.normalize(p)

Normalize a string path, taking care of '..' and '.' parts.

2

path.join([path1][, path2][, ...])

Join all the arguments together and normalize the resulting path.

3

path.resolve([from ...], to)

Resolves to an absolute path.

4

path.isAbsolute(path)

Determines whether the path is an absolute path. An absolute path will always resolve to the same location, regardless of the working directory.

5

path.relative(from, to)

Solve the relative path from from to to.

6

path.dirname(p)

Return the directory name of a path. Similar to the Unix dirname command.

7

path.basename(p[, ext])

Return the last portion of a path. Similar to the Unix basename command.

8

path.extname(p)

Return the extension of the path, from the last '.' to end of string in the last portion of the path. If there is no '.' in the last portion of the path or the first character of it is '.', then it returns an empty string.

9

path.parse(pathString)

Returns an object from a path string.

10

path.format(pathObject)

Returns a path string from an object, the opposite of path.parse above.