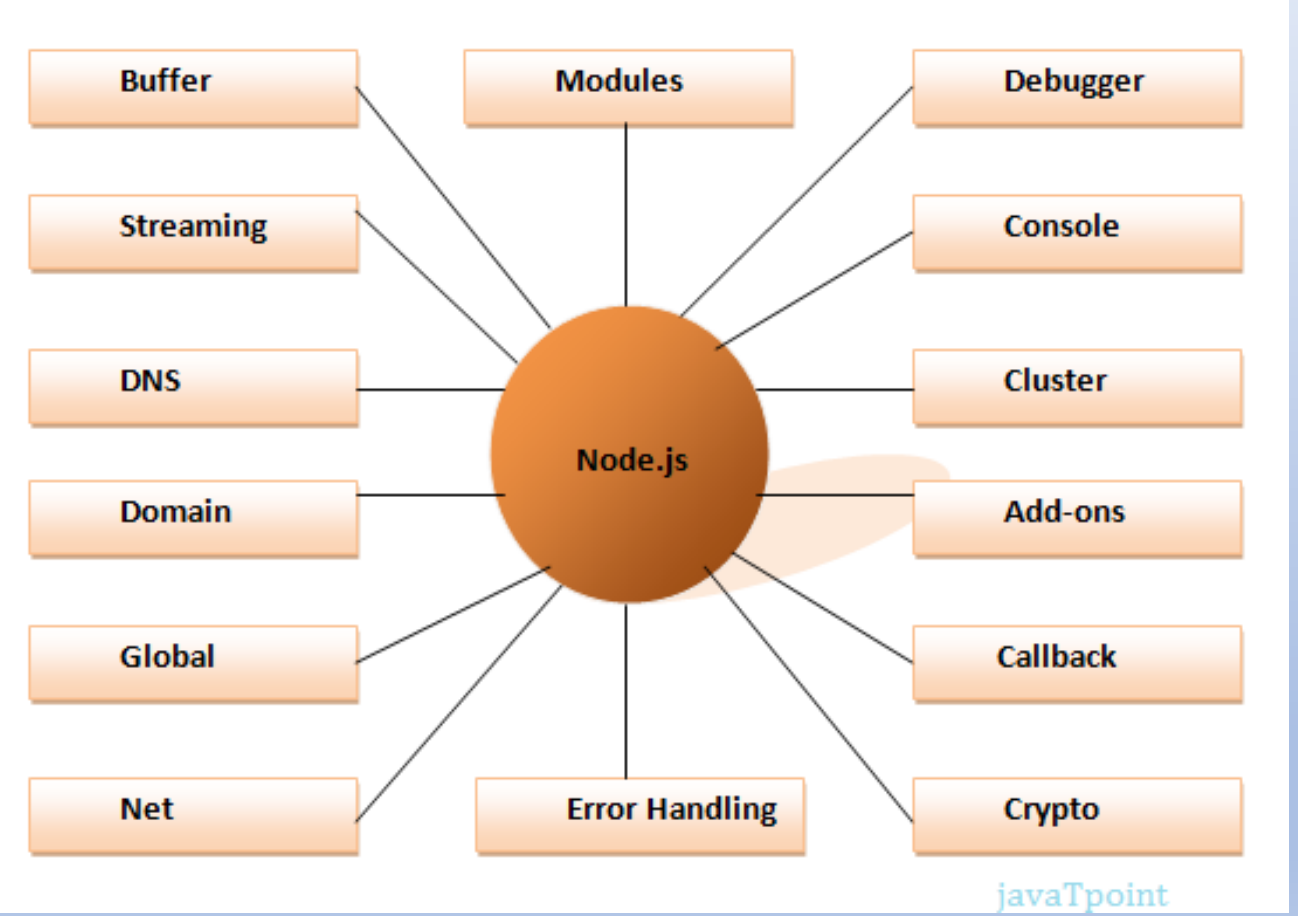
NodeJS

NodeJS

- 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.
- When Node.js performs an I/O operation, like reading from the network, accessing a database or the filesystem, instead of blocking the thread and wasting CPU cycles waiting, Node.js will resume the operations when the response comes back.
- This allows Node.js to handle thousands of concurrent connections with a single server without introducing the burden of managing thread concurrency, which could be a significant source of bugs.
- Node.js has a unique advantage because millions of frontend developers that write JavaScript for the browser are now able to write the server-side code in addition to the client-side code without the need to learn a completely different language.
- In Node.js the new ECMAScript standards can be used without problems, as you don't have to wait for all your users to update their browsers - you are in charge of deciding which ECMAScript version to use by changing the Node.js version, and you can also enable specific experimental features by running Node.js with flags.

NodeJS

Important parts of Node.js



NodeJS

Features of Node.js

1. **Extremely fast:** Node.js is built on Google Chrome's V8 JavaScript Engine, so its library is very fast in code execution.
2. **I/O is Asynchronous and Event Driven:** All APIs of Node.js library are asynchronous i.e. non-blocking. So a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a notification mechanism of Events of Node.js helps the server to get a response from the previous API call. It is also a reason that it is very fast.
3. **Single threaded:** Node.js follows a single threaded model with event looping.
4. **Highly Scalable:** Node.js is highly scalable because event mechanism helps the server to respond in a non-blocking way.
5. **No buffering:** Node.js cuts down the overall processing time while uploading audio and video files. Node.js applications never buffer any data. These applications simply output the data in chunks.
6. **Open source:** Node.js has an open source community which has produced many excellent modules to add additional capabilities to Node.js applications.
7. **License:** Node.js is released under the MIT license.

NodeJS

Installation of Node.js

- **Verify Installation:**
    1. Create a file with <name>.js extension
    2. Add the line: console.log("Hello, World!")
    3. Execute with below command:

        node file_name.js
    4. You should get the output: Hello, World!

NodeJS

What is front-end and back-end development using JS

Front End and Back End: Frontend and Backend are the two most popular terms used in web development. These terms are very crucial for web development but are quite different from each other. Each side needs to communicate and operate effectively with the other as a single unit to improve the website’s functionality.

Front End Development

Front End Development: The part of a website that the user interacts with directly is termed the front end. It is also referred to as the ‘client side’ of the application. It includes everything that users experience directly: text colors and styles, images, graphs and tables, buttons, colors, and navigation menu. HTML, CSS, and JavaScript are the languages used for Front End development. The structure, design, behavior, and content of everything seen on browser screens when websites, web applications, or mobile apps are opened up, is implemented by front End developers. Responsiveness and performance are two main objectives of the Front End. The developer must ensure that the site is responsive i.e. it appears correctly on devices of all sizes no part of the website should behave abnormally irrespective of the size of the screen.

Front End Development

Front end Languages: The front end portion is built by using some languages which are discussed below:

- **HTML:** HTML stands for Hypertext Markup Language. It is used to design the front-end portion of web pages using a markup language. HTML is the combination of Hypertext and Markup language. Hypertext defines the link between the web pages. The markup language is used to define the text documentation within the tag which defines the structure of web pages.
- **CSS:** Cascading Style Sheets fondly referred to as CSS is a simply designed language intended to simplify the process of making web pages presentable. CSS allows you to apply styles to web pages. More importantly, CSS enables you to do this independent of the HTML that makes up each web page.
- **JavaScript:** JavaScript is a famous scripting language used to create magic on the sites to make the site interactive for the user. It is used to enhancing the functionality of a website to running cool games and web-based software.

There are many other languages through which one can do front-end development depending upon the framework

for example *Flutter* user *Dart*, *React* uses *JavaScript* and *Django* uses *Python*, and much more.

Front End Development

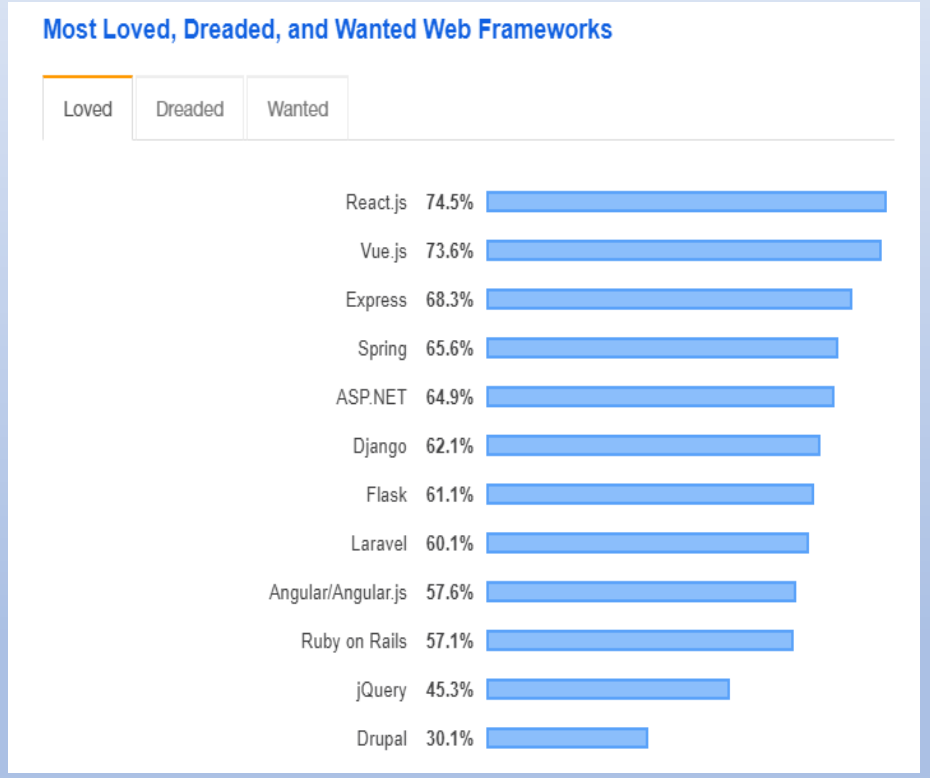
Front End Frameworks and Libraries:

- **AngularJS:** AngularJs is a JavaScript open-source front-end framework that is mainly used to develop single-page web applications(SPAs). It is a continuously growing and expanding framework which provides better ways for developing web applications. It changes the static HTML to dynamic HTML. It is an open-source project which can be free. It extends HTML attributes with Directives, and data is bound with HTML.
- **React.js:** React is a declarative, efficient, and flexible JavaScript library for building user interfaces. ReactJS is an open-source, component-based front-end library responsible only for the view layer of the application. It is maintained by Facebook.
- **Bootstrap**: Bootstrap is a free and open-source tool collection for creating responsive websites and web applications. It is the most popular HTML, CSS, and JavaScript framework for developing responsive, mobile-first websites.
- **jQuery:** jQuery is an open-source JavaScript library that simplifies the interactions between an HTML/CSS document, or more precisely the Document Object Model (DOM), and JavaScript. Elaborating the terms, jQuery simplifies HTML document traversing and manipulation, browser event handling, DOM animations, Ajax interactions, and cross-browser JavaScript development.
- **SASS:** It is the most reliable, mature, and robust CSS extension language. It is used to extend the functionality of an existing CSS of a site including everything from variables, inheritance, and nesting with ease.
- **Flutter:** Flutter is an open-source UI development SDK managed by google. It is powered by Dart programming language. It builds performant and good-looking natively complied applications for mobile (Ios, Android), web, and desktop from a single code base. The key selling point of flutter is flat development is made easier, expressive, and flexible UI and native performance. In march 2021 flutter announce Flutter 2 which upgrades flutter to build release applications for the web, and the desktop is in beta state.
- Some other libraries and frameworks are Semantic-UI, Foundation, Materialize, Backbone.js, Ember.js, etc.

Difference

Angular vs ReactJS

- JavaScript is one of the most popular languages among developers nowadays. There are a lot of developers, freshers, and experienced love to build their application or project using JavaScript but still, there is confusion when they have to pick up right framework or library for their project.
  Angular and ReactJs is their topmost priority but still, most of them are unable to decide which one would be good for their project. Freshers want to know which one is easy to learn and which one has more demand in the market for job purposes. We need to keep in mind ReactJS is a library to build interactive user-interfaces, on the other hand Angular is a complete framework



Difference

Angular vs ReactJS

Before we come to any conclusion we need to keep in mind that there is no best framework or library. Choosing a framework or library completely depends on your project level, requirements, and your goals. Every framework or library has some pros and cons, same with React and Angular. From the above all factors if you are a beginner or have less coding practice also if you want stability for your project you can go with React because its learning curve is fast and easier also job in the market is higher than Angular. It might be frustrating if you are choosing Angular because after every 6 months you will experience major upgrades for Angular. Another thing is if you want a full-blown framework to build a large scale project and love to follow straight forward coding strategy then go with Angular.

Back End Development

Backend is the server-side of the website. It stores and arranges data, and also makes sure everything on the client-side of the website works fine. It is the part of the website that you cannot see and interact with. It is the portion of software that does not come in direct contact with the users. The parts and characteristics developed by backend designers are indirectly accessed by users through a front-end application. Activities, like writing APIs, creating libraries, and working with system components without user interfaces or even systems of scientific programming, are also included in the backend.

Back End Development

Back end Languages: The back end portion is built by using some languages which are discussed below:

- **PHP:** PHP is a server-side scripting language designed specifically for web development. Since PHP code executed on the server-side, so it is called a server-side scripting language.
- **C++:** It is a general-purpose programming language and widely used nowadays for competitive programming. It is also used as a backend language.
- **Java:** Java is one of the most popular and widely used programming languages and platforms. It is highly scalable. Java components are easily available.
- **Python:** Python is a programming language that lets you work quickly and integrate systems more efficiently.
- **JavaScript:** JavaScript can be used as both (front end and back end) programming languages.
- **Node.js:** Node.js is an open-source and cross-platform runtime environment for executing JavaScript code outside a browser. You need to remember that NodeJS is not a framework, and it's not a programming language. Most people are confused and understand it's a framework or a programming language. We often use Node.js for building back-end services like APIs like Web App or Mobile App. It's used in production by large companies such as Paypal, Uber, Netflix, Wallmart, and so on.

Back End Frameworks:

- The list of back-end frameworks are: Express, Django, Rails, Laravel, Spring, etc.
- The other back-end program/scripting languages are C#, Ruby, REST, GO, etc.

Difference

Difference between Frontend and Backend: Frontend and backend development are quite different from each other, but still, they are two aspects of the same situation. The frontend is what users see and interact with and the backend is how everything works.

- The frontend is the part of the website users can see and interact with such as the graphical user interface (GUI) and the command line including the design, navigating menus, texts, images, videos, etc. Backend, on the contrary, is the part of the website users cannot see and interact with.
- The visual aspects of the website that can be seen and experienced by users are frontend. On the other hand, everything that happens in the background can be attributed to the backend.
- Languages used for the front end are HTML, CSS, JavaScript while those used for the backend include Java, Ruby, Python, .Net.

Backend Framework

Backend Framework:

- The two back-end web frameworks i.e. **Laravel**, **Django**, and the run-time environment **NodeJS** helps in the development activities. All end up acquiring the same objective, that's develop a Web Application. What leads to the comparison is:
- **Laravel:** It is a PHP framework that is free and open-source enabling developers to use the pattern of MVC in the development needs.
- **NodeJS:** It is a JavaScript Runtime Environment that is used for Cross-Platform development needs.
- **Django:** It is a Python-based framework that allows developers to use a systematic approach for the Web development process.

Backend Framework

Laravel: It was released years after Django was developed and it is created by Taylor and Otwell, in order to use Laravel it is important for the developers to have knowledge about PHP basics. Laravel has in-built features that make the development process easy hence reducing the development time and most of the applications which come under content management system use Laravel. If you are working on a new website from scratch then Laravel is the powerful feature that helps you in all the phases for Web development.

Advantage:

- It is an excellent choice of framework for PHP.
- It is based on MVC so it eliminates the need for writing HTML codes.
- It provides easy integration of logic within the website using a blade template engine.
- It has built-in Authorization & Authentication System and also Easy Integration with Mail System.
- Provides Smooth Automation of testing work.

Disadvantages:

- It does not have inbuilt tools and requires third-party integration for custom website development.
- It is pretty slow and the developers need to be adept in PHP before working on Laravel.

Backend Framework

NodeJS: Talking about Node JS, it is not a framework but a server also. Based on JS, it embeds all the features above, meaning that no more multi-threading and not meant for the beginners. So, NodeJS is a fundamental sense of a JS server that primarily acts as server-side browsing. It is open-source and eases the development of cross-platform web applications. The primary reason why developers like working on Node JS is the fact that it works on a single thread. The entire server is event-based and causes on receiving callbacks. This enables the server to come back every time it is called and prevents it from being a pause or in a sleep state.

Advantages:

- The performance of an app developed using NodeJS is higher than others.
- It comes along with an excellent package manager.
- NodeJS has extended support in the form of libraries.
- Works best when you need to build APIs.
- It provides quick and easy handling of users concurrent requests.

Disadvantages:

- The fact that node.js involves asynchronous programming, not all developers find it easy to understand and could be difficult to work with.
- Callbacks lead to tons of nested callbacks.

Backend Framework

Django: Before 2005, No one has thought we can have a web development framework based on python. And now Django is the heart and soul for many of the developers out there. Instagram, Mozilla, Bitbucket, you will see that Django is used for the developments of Web applications and the framework is light weighted and it really has plenty of features to be developed and deployed on Web applications.

Advantages:

- It has an easy learning curve.
- Seamless collaboration with relational databases.
- It is Backed up by huge support from the user community.
- It has High Scalability.
- It is detailed and crisp documentation.

Disadvantages:

- It is in face monolithic, which means it is a single-tiered software application.
- It does not work well with small-scale apps.
- Geeks should have expertise in the language before implementing it.

Backend Framework

Field-wise comparison:

- **Scalability & Performance:** Node.js ranks high in performance, Django has its way being scalable with Laravel having a set of features that can keep your website one step ahead in the market.
- **Architecture:** Django has an MVT architecture where Laravel follows an MVC pattern. On the other hand, the node is event-driven.
- **Security:** Django is the best when it comes to security with Laravel next. However, despite the fact that the node is pretty famous, it could have holes and remain unnoticed for a longer time period.
- **Customizability:** Being backed by JavaScript, node.js has the maximum customization options, whereas Django calls for a lot more complexities when one needs customization. Laravel, on the contrary, needs third party tools to add and personalize the website.
- **Verdict:** We have seen all three separately and in conjunction. Now, which one should you choose and opt for depends on your specific requirements as we have developers for both the technologies and after all it depends on your project requirements. Remember, one might be better than the other, but the decision has to be made on which one maps your needs best.

Backend Framework

Difference Between Django and Node.js:

Django	NodeJS
It is an open-source Python-based web framework to design web applications open-source.	It is an open-source and JS runtime environment to develop web applications.
Django is programmed in Python.	Node.js is written in C, C++, and JavaScript.
Django is less scalable for small apps.	Node.js is more scalable than Django for small apps.
Django follows Model template View architecture.	Node.js follows event-driven programming.
Django is more complex than node.js.	Node.js is less complex than Django.
It is modern and behind Node.js in utilization.	It is utilized broadly in numerous nations and ahead comparatively.
Django web development is more stable than node.js.	Node.js web development is very less stable than Django.

NPM

npm(node package manager) is regarded as the standard package manager used in javascript. The npm registry crossed a million packages last year(Jun ’19). It is the largest single-language code repository. That being said, it is a little obvious now that **npm is a big deal**.

npm is automatically installed when you install Node.js

Since its a package manager, it is used to manage downloads and handle dependencies of your project.

It is a common saying that

There is a package for everything in npm

Well, almost everything. There’s always more.

NPM

npm is written entirely in JavaScript(JS) and was developed by Isaac Schlueter. It was initially used to download and manage dependencies, but it has since also used frequently in frontend JavaScript.

npm can manage packages that are local dependencies of a particular project, as well as globally-installed JavaScript tools. In addition to plain downloads, **npm** also manages versioning, so you can install any version, higher or lower according to the needs of your project. If no version is mentioned, the latest version of the package is installed.

NPM

How to use

→ If package.json file exists in your project directory, all you need to do is use this command - npm install

This command will initialize the node_modules folder and install all packages that the project needs.

And if you need to update the installed packages, - npm update

All packages will be updated to their latest versions.

→ If you just need to install a single package, you can use this command - npm install <package_name>

Similarly, if you just need to update a single package, all you need to do is - npm update <package_name>

Note: By default, packages are installed in the local scope. If you need to install the package at global scope, you need to mention the flag –g : npm install <package_name> -g

This will install the package at the global scope in the system directory.

NPM Module

Node.js modules are a type of package that can be published to npm.

How to publish your own package
Several developers feel the need to use some functionality of one project in another. Normally, the developer copies code from one project and pastes in another but if it is common functionality that may be used in several projects its a better and good practice to publish your reusable codes as npm packages.

1. npm init
When you are in your desired directory, open the CLI, and use this command: npm init
A text utility appears and you are asked to enter a few details to create a basic package.json file such as package name, version, description, author, etc.
Enter the details of the package as you please. The defaults will be mentioned in parentheses like this →
package name: (test-pkg)
version: (1.0.0)
description: This is a test package
.
.
.
Just hit Enter if you want the default options.

NPM Module

A basic package.json will be created looking something like this →

```
{
  "name": "test-pkg",
  "version": "1.0.0",
  "description": "This is a test package",
  "main": "index.js",
  "author": "after-academy",
  "license": "ISC"
}
```

Obviously, you can change the package.json file later if you want.

NPM Module

2. Source

Now, you need to prepare the source code. If the code is not too big, it is typically just write in the main file(index.js here). Else, conventionally a src directory is used where your abstract code is stored in several files.

★ Remember to export the code using module.exports

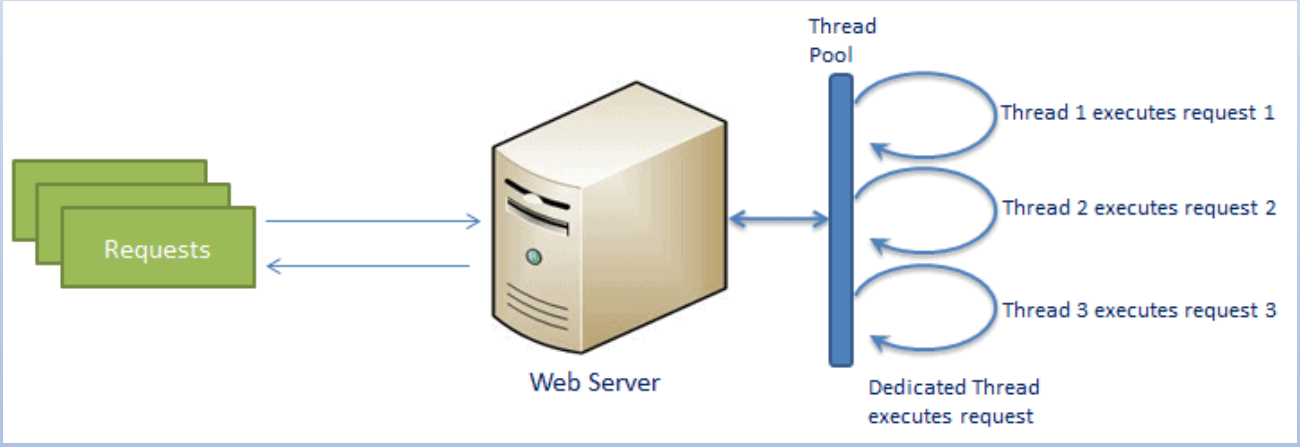
3. Test

You now need to thoroughly test your code before publishing. This is how you, as a developer, confirm that your package can actually be used.

Node.js Process Model

Traditional Web Server 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.



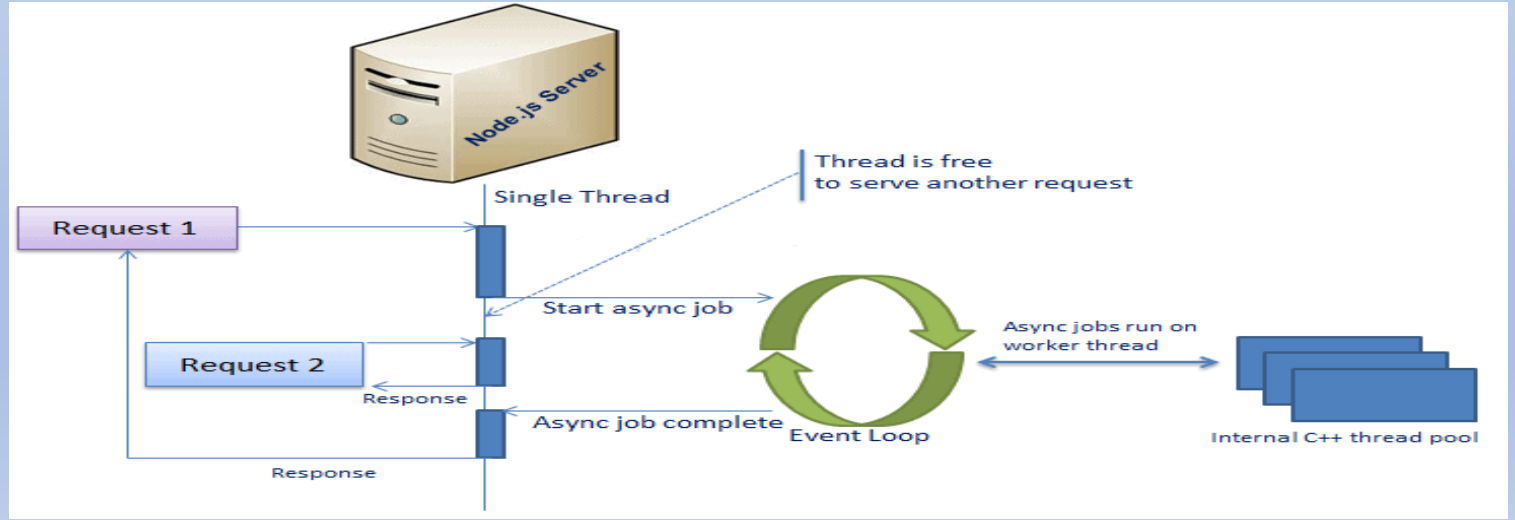
Node.js Process Model

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.

An event loop is constantly watching for the events to be raised for an asynchronous job and executing callback function when the job completes. Internally, Node.js uses libev for the event loop which in turn uses internal C++ thread pool to provide asynchronous I/O.

Libev: A full-featured and high-performance (see benchmark) event loop that is loosely modelled after libevent, but without its limitations and bugs. It is used in GNU Virtual Private Ethernet, rxvt-unicode, auditd, the Deliantra MORPG Server and Client, and many other programs.

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.



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 Module Types

Node.js includes three types of modules:

- Core Modules
- Local Modules
- Third Party Modules

Node.js Core 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.

The following table lists some of the important core modules in Node.js.

Node.js Module

Core Module	Description
http	http module includes classes, methods and events to create Node.js http server.
url	url module includes methods for URL resolution and parsing.
querystring	querystring module includes methods to deal with query string.
path	path module includes methods to deal with file paths.
fs	fs module includes classes, methods, and events to work with file I/O.
util	util module includes utility functions useful for programmers.

Node.js Module

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.

```
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.

Node.js Module

```
http.createServer(function (request, response) {
   // Send the HTTP header
   // HTTP Status: 200 : OK
   // Content Type: text/plain
request.
   response.writeHead(200, {'Content-Type': 'text/plain'});

   // Send the response body as "Hello World"
   response.end('Hello World\n');
}).listen(8081);

// Console will print the message
console.log('Server running at http://127.0.0.1:8081/');
```

Node.js Module

Node.js Local Module

Local modules are modules created locally in your Node.js application. These modules include different functionalities of your application in separate files and folders. You can also package it and distribute it via NPM, so that Node.js community can use it. For example, if you need to connect to MongoDB and fetch data then you can create a module for it, which can be reused in your application.

Writing Simple Module

Let's write simple logging module which logs the information, warning or error to the console.

In Node.js, module should be placed in a separate JavaScript file. So, create a Log.js file and write the following code in it.

```
var log = {
        info: function (info) {
            console.log('Info: ' + info);
        },
        warning:function (warning) {
            console.log('Warning: ' + warning);
        },
        error:function (error) {
            console.log('Error: ' + error);
        }
    };
module.exports = log
```

In the above example of logging module, we have created an object with three functions - info(), warning() and error(). At the end, we have assigned this object to module.exports. The module.exports in the above example exposes a log object as a module.

Node.js Module

The module.exports is a special object which is included in every JS file in the Node.js application by default. Use module.exports or exports to expose a function, object or variable as a module in Node.js.

Loading Local Module

To use local modules in your application, you need to load it using require() function in the same way as core module. However, you need to specify the path of JavaScript file of the module.

The following example demonstrates how to use the above logging module contained in Log.js.

app.js

```
var myLogModule = require('./Log.js');
myLogModule.info('Node.js started');
```

In the above example, app.js is using log module. First, it loads the logging module using require() function and specified path where logging module is stored. Logging module is contained in Log.js file in the root folder. So, we have specified the path './Log.js' in the require() function. The '.' denotes a root folder.

The require() function returns a log object because logging module exposes an object in Log.js using module.exports. So now you can use logging module as an object and call any of its function using dot notation e.g myLogModule.info() or myLogModule.warning() or myLogModule.error()

```
C:\> node app.js
Info: Node.js started
```

Thus, you can create a local module using module.exports and use it in your application.

Node.js Module

Export Module in Node.js
Here, you will learn how to expose different types as a module using module.exports.

The module.exports is a special object which is included in every JavaScript file in the Node.js application by default. The module is a variable that represents the current module, and exports is an object that will be exposed as a module. So, whatever you assign to module.exports will be exposed as a module.

Let's see how to expose different types as a module using module.exports.

Node.js Module

Export Literals
As mentioned above, exports is an object. So it exposes whatever you assigned to it as a module. For example, if you assign a string literal then it will expose that string literal as a module.

The following example exposes simple string message as a module in Message.js.

Message.js
```
module.exports = 'Hello world';
```

Now, import this message module and use it as shown below.

app.js
```
var msg = require('./Messages.js');
console.log(msg);
```

Run the above example and see the result, as shown below.

```
C:\> node app.js
Hello World
```

Note: You must specify ./ as a path of root folder to import a local module. However, you do not need to specify the path to import Node.js core modules or NPM modules in the require() function.

Node.js Module

Export Object
The exports is an object. So, you can attach properties or methods to it. The following example exposes an object with a string property in Message.js file.

Message.js
```
exports.SimpleMessage = 'Hello world';
//or
module.exports.SimpleMessage = 'Hello world';
```
In the above example, we have attached a property SimpleMessage to the exports object. Now, import and use this module, as shown below.

app.js
```
var msg = require('./Messages.js');
console.log(msg.SimpleMessage);
```

In the above example, the require() function will return an object { SimpleMessage : 'Hello World'} and assign it to the msg variable. So, now you can use msg.SimpleMessage.

Run the above example by writing node app.js in the command prompt and see the output as shown below.

```
C:\> node app.js
Hello World
```

Node.js Module

In the same way as above, you can expose an object with function. The following example exposes an object with the log function as a module.

Log.js

```
module.exports.log = function (msg) {
    console.log(msg);
};
```

The above module will expose an object- { log : function(msg){ console.log(msg); } } . Use the above module as shown below.

app.js

```
var msg = require('./Log.js');

msg.log('Hello World');
```

Run and see the output in command prompt as shown below.

```
C:\> node app.js
Hello World
```

You can also attach an object to module.exports, as shown below.

data.js Copy

```
module.exports = {
    firstName: 'James',
    lastName: 'Bond'
}
```

app.js Copy

```
var person = require('./data.js');
```

Node.js Module

Export Function
You can attach an anonymous function to exports object as shown below.

Log.js
```
module.exports = function (msg) {
    console.log(msg);
};
```
Now, you can use the above module, as shown below.

app.js
```
var msg = require('./Log.js');

msg('Hello World');
```
The msg variable becomes a function expression in the above example. So, you can invoke the function using parenthesis (). Run the above example and see the output as shown below.

```
C:\> node app.js
Hello World
```

Node.js Module

Export Function as a Class
In JavaScript, a function can be treated like a class. The following example exposes a function that can be used like a class.

Person.js
```
module.exports = function (firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.fullName = function () {
        return this.firstName + ' ' + this.lastName;
    }
}
```
The above module can be used, as shown below.

app.js
```
var person = require('./Person.js');

var person1 = new person('James', 'Bond');

console.log(person1.fullName());
```
As you can see, we have created a person object using the new keyword. Run the above example, as shown below.

```
C:\> node app.js
James Bond
```
In this way, you can export and import a local module created in a separate file under root folder.

Node.js Module

Node.js also allows you to create modules in sub folders. Let's see how to load module from sub folders.

Load Module from the Separate Folder

Use the full path of a module file where you have exported it using module.exports. For example, if the log module in the log.js is stored under the utility folder under the root folder of your application, then import it, as shown below.

app.js

```
var log = require('./utility/log.js');
```

In the above example, . is for the root folder, and then specify the exact path of your module file. Node.js also allows us to specify the path to the folder without specifying the file name. For example, you can specify only the utility folder without specifying log.js, as shown below.

app.js

```
var log = require('./utility');
```

In the above example, Node.js will search for a package definition file called package.json inside the utility folder. This is because Node assumes that this folder is a package and will try to look for a package definition. The package.json file should be in a module directory. The package.json under utility folder specifies the file name using the main key, as shown below.

./utility/package.json

```
{
    "name" : "log",
    "main" : "./log.js"
}
```

Now, Node.js will find the log.js file using the main entry in package.json and import it.

Note: If the package.json file does not exist, then it will look for index.js file as a module file by default.

Node.js – Parse URL

Node.js Parse URL : Split a URL into readable parts and extract search parameters using built-in Node.js URL module.

To parse URL in Node.js : use url module, and with the help of parse and query functions, you can extract all the components of URL.

Steps – Parse URL components in Node.js
Following is a step-by-step guide to program on how to parse URL into
readable parts in Node.js.

Step 1: Include URL module

```
var url = require('url');
```

Step 2: Take URL to a variable

Following is a sample URL that we shall parse.

```
var address = 'http://localhost:8080/index.php?type=page&action=update&id=5221';
```

Ex: https://github.com/amitkumar1211/MERN-training

Step 3: Parse URL using parse function.

```
var q = url.parse(address, true);
```

Node.js – Parse URL

Step 4: Extract HOST, PATHNAME and SEARCH string using dot operator.

q.host
q.pathname
q.Search

Step 5: Parse URL Search Parameters using query function.

var qdata = q.query;

Step 6: Access Search Parameters

qdata.type
qdata.action
qdata.id

Node.js – Parse URL

Example 1 –
urlParsingExample.js

```
// include url module
var url = require('url');
var address = 'http://localhost:8080/index.php?type=page&action=update&id=5221';
var q = url.parse(address, true);
console.log(q.host); //returns 'localhost:8080'
console.log(q.pathname); //returns '/index.php'
console.log(q.search); //returns '?type=page&action=update&id=5221'
var qdata = q.query; // returns an object: { type: page, action: 'update',id='5221' }
console.log(qdata.type); //returns 'page'
console.log(qdata.action); //returns 'update'
console.log(qdata.id); //returns '5221'
```

Output

```
$ node urlParsingExample.js
localhost:8080
/index.php
?type=page&action=update&id=5221
page
update
5221
```

Node.js Web Server

The web server will handle all the http requests for the web application e.g IIS is a web server for ASP.NET web applications and Apache is a web server for PHP or Java web applications.

Node.js provides capabilities to create your own web server which will handle HTTP requests asynchronously. You can use IIS or Apache to run Node.js web application but it is recommended to use Node.js web server.

Create Node.js Web Server
Node.js makes it easy to create a simple web server that processes incoming requests asynchronously.

The following example is a simple Node.js web server contained in server.js file.

server.js

```
var http = require('http'); // 1 - Import Node.js core module

var server = http.createServer(function (req, res) {   // 2 - creating server

    //handle incomming requests here..
});

server.listen(5000); //3 - listen for any incoming requests
console.log('Node.js web server at port 5000 is running..')
```

Node.js Web Server

In the example, we import the http module using require() function. The http module is a core module of Node.js, so no need to install it using NPM. The next step is to call createServer() method of http and specify callback function with request and response parameter. Finally, call listen() method of server object which was returned from createServer() method with port number, to start listening to incoming requests on port 5000. You can specify any unused port here.

Run the above web server by writing node server.js command in command prompt or terminal window and it will display message as shown below.

```
C:\> node server.js
Node.js web server at port 5000 is running..
```

This is how you create a Node.js web server using simple steps. Now, let's see how to handle HTTP request and send response in Node.js web server.

Handle HTTP Request

The http.createServer() method includes request and response parameters which is supplied by Node.js. The request object can be used to get information about the current HTTP request e.g., url, request header, and data. The response object can be used to send a response for a current HTTP request.

The following example demonstrates handling HTTP request and response in Node.js.

Node.js Web Server

```
var http = require('http'); // Import Node.js core module
var server = http.createServer(function (req, res) {   //create web server
    if (req.url == '/') { //check the URL of the current request
        // set response header
        res.writeHead(200, { 'Content-Type': 'text/html' });

        // set response content
        res.write('<html><body><p>This is home Page.</p></body></html>');
        res.end();

    }
    else if (req.url == "/student") {
        res.writeHead(200, { 'Content-Type': 'text/html' });
        res.write('<html><body><p>This is student Page.</p></body></html>');
        res.end();
}
    else if (req.url == "/admin") {
res.writeHead(200, { 'Content-Type': 'text/html' });
        res.write('<html><body><p>This is admin Page.</p></body></html>');
        res.end();
}
    else
        res.end('Invalid Request!');
});
server.listen(5000); //6 - listen for any incoming requests
console.log('Node.js web server at port 5000 is running..')
```

Node.js Web Server

In the above example, req.url is used to check the url of the current request and based on that it sends the response. To send a response, first it sets the response header using writeHead() method and then writes a string as a response body using write() method. Finally, Node.js web server sends the response using end() method.

Now, run the above web server as shown below.

```
C:\> node server.js
Node.js web server at port 5000 is running..
```

To test it, you can use the command-line program curl, which most Mac and Linux machines have pre-installed.

```
curl -i http://localhost:5000
```

Output:

```
HTTP/1.1 200 OK
Content-Type: text/plain
Date: Tue, 8 Sep 2015 03:05:08 GMT
Connection: keep-alive
This is home page.
```

Node.js Web Server

Sending JSON Response

The following example demonstrates how to serve JSON response from the Node.js web server.

server.js

```
var http = require('http');

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

    if (req.url == '/data') { //check the URL of the current request
            res.writeHead(200, { 'Content-Type': 'application/json' });
            res.write(JSON.stringify({ message: "Hello World"}));
            res.end();
    }
});

server.listen(5000);

console.log('Node.js web server at port 5000 is running..')
```

So, this way you can create a simple web server that serves different responses.

References:

Reference:

https://nodejs.org/en/

https://requirejs.org/docs/commonjs.html