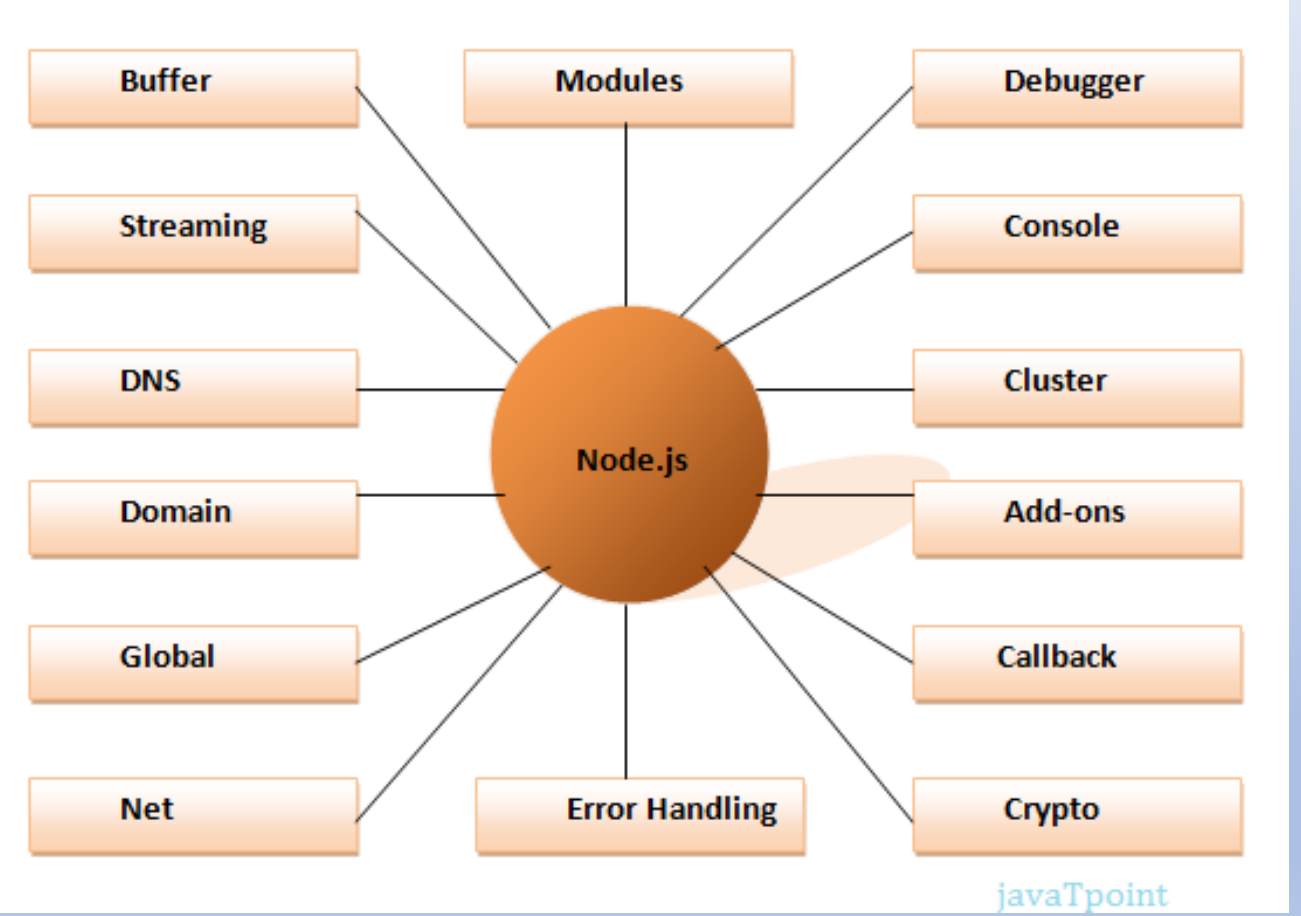
NodeJS

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

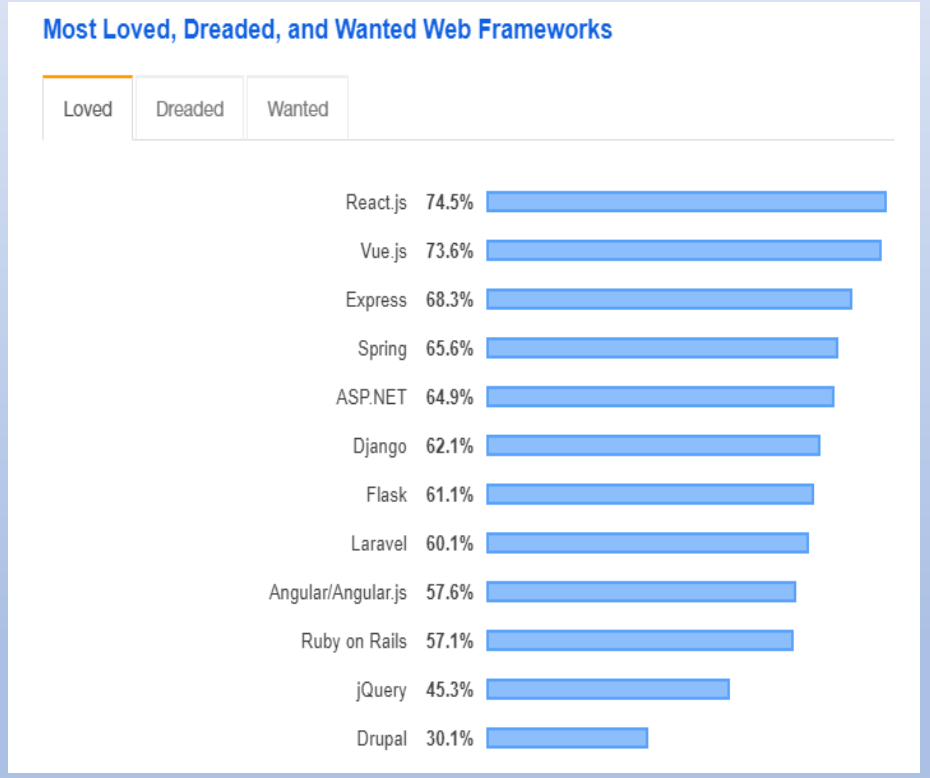
Front End Frameworks and Libraries:

- **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

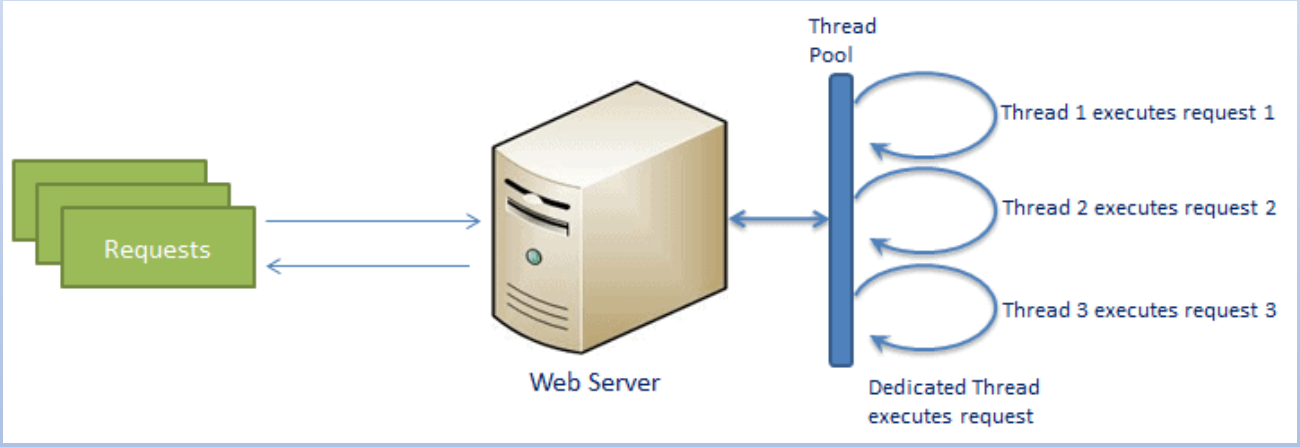
3. Test

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.



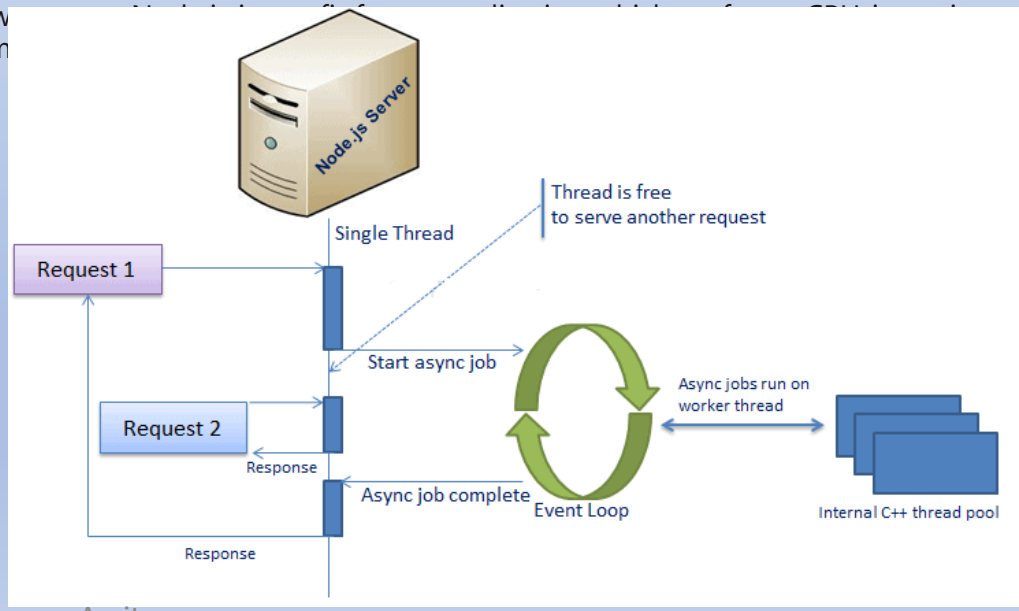
Node.js Process Model

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 [illegible] operations like image processing or other heavy computation work because it takes tim



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

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';
```

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.

Node.js Redirect URL

Node.js Redirect URL :

A redirect could be applied in situations like :

Some of the resources are moved permanently to a new location and you want to redirect your users to the new location of moved resources.
Some of the pages in your web application are removed, and when a request comes for that page, you would like your users be redirected to home page or some custom page.
There are three main types of HTTP redirects. Please refer wiki page for redirects[https://en.wikipedia.org/wiki/List_of_HTTP_status_codes#3xx_Redirection].

But remember that the HTTP redirect code (say 301, 302, 307, etc.,) affect the page ranking for the original or redirected url, and each of the redirect codes affect differently. For example, if you have moved the resource permanently, using 301 HTTP code in the response passes the juice to the redirected URL, while 302 or 307 does not.

For the following examples, consider that there are two pages : page-a.html and page-b.html, that your web application serves. And we have a 404_not_found.html to be displayed when a requested resource is not present.

Node.js Redirect URL

node-js-http-redirect.js

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

// create a http server
http.createServer(function (req, res) {

    if (req.url == '/page-c.html') {
        // redirect to page-b.html with 301 (Moved Permanently) HTTP code in the response
        res.writeHead(301, { "Location": "http://" + req.headers['host'] + '/page-b.html' });
        return res.end();
    } else {
        // for other URLs, try responding with the page
        console.log(req.url)
        // read requested file
        fs.readFile(req.url.substring(1),
            function(err, data) {
                if (err) throw err;
                res.writeHead(200);
                res.write(data.toString('utf8'));
                return res.end();
        });
    }
}).listen(8085);
```

Node.js Redirect URL

node-js-http-redirect-file-not-found.js

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

// create a http server
http.createServer(function (req, res) {
    var filePath = req.url.substring(1);
    fs.readFile(filePath,
        function(err, data) {
            // if there is an error reading the file, redirect it to page-b.html
            if (err){
                // redirect to page-b.html with 302 HTTP code in response
                res.writeHead(302, { "Location": "http://" + req.headers['host'] + '/page-b.html' });
                return res.end();
            }
            res.writeHead(200);
            res.write(data.toString('utf8'));
            return res.end();
    });
}).listen(8085);
```

Node FS File Operations

With Node FS, you may do following actions on files :

Reading files
Creating or Overwriting files
Updating files
Deleting files
Renaming files
Include Node.js FS Module
To include Node.js FS module in Node.js program, use the following require() statement.

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

Steps to Read File
Following is a step by step guide to read content of a File in Node.js :

Step 1 : Include File System built-in module to your Node.js program.

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

Step 2 : Read file using readFile() function.

```
fs.readFile('<fileName>',<callbackFunction>)
```

Callback function is provided as an argument to readFile function. When reading the file is completed (could be with or without error), call back function is called with err(if there is an error reading file) and data(if reading file is successful).

Step 3 : Create a sample file, say sample.html with some content in it. Place the sample file at the location of node.js example program, which is provided below.

Node FS File Operations

readFileExample.js

```
// include file system module
var fs = require('fs');
// read file sample.html
fs.readFile('sample.html',
    // callback function that is called when reading file is done
    function(err, data) {
        if (err) throw err;
        // data is a buffer containing file content
        console.log(data.toString('utf8'))
});
```

Syntax – writeFile() function

```
fs.writeFile('<fileName>',<content>, callbackFunction)
```

A new file is created with the specified name. After writing to the file is completed (could be with or without error), callback function is called with error if there is an error reading file.

If a file already exists with the name, the file gets overwritten with a new file. Care has to be taken while using this function, as it overwrites existing file.

Syntax – appendFile() function

```
fs.appendFile('<fileName>',<content>, callbackFunction)
```

If the file specified in the appendFile() function does not exist, a new file is created with the content passed to the function

Node FS File Operations

Syntax – open() function

```
fs.open('<fileName>',<file_open_mode>, callbackFunction)
```

If the specified file is not found, a new file is created with the specified name and mode and sent to the callback function.

Steps – Create a File in Node.js

Step 1 : Include File System built-in module to your Node.js program

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

Step 2 : Create file using one the methods mentioned above. Following are the examples demonstrating to create a file in Node.js with each of these methods.

Example 1 – Create File using writeFile()
createFileExample.js

```
// include node fs module
var fs = require('fs');

// writeFile function with filename, content and callback function
fs.writeFile('newfile.txt', 'Learn Node FS module', function (err) {
  if (err) throw err;
  console.log('File is created successfully.');
});
```

Node FS File Operations

Example 2 – Create File using appendFile()

createFileExample2.js

```
// include node fs module
var fs = require('fs');

// appendFile function with filename, content and callback function
fs.appendFile('newfile_2.txt', 'Learn Node FS module', function (err) {
  if (err) throw err;
  console.log('File is created successfully.');
});
```

Example 3 – Create File using open()

createFileExample3.js

```
// include node fs module
var fs = require('fs');

// open function with filename, file opening mode and callback function
fs.open('newfile_3.txt', 'w', function (err, file) {
  if (err) throw err;
  console.log('File is opened in write mode.');
});
```

Node FS File Operations

Write to File
We can write data to file in Node.js using fs module.

syntax of writeFile() function is

```
fs = require('fs');
fs.writeFile(filename, data, [encoding], [callback_function])
where
```

filename : [mandatory] name of the file, the data has to be written to
data : [mandatory] content that has to be written to file
encoding : [optional] encoding standard that has to be followed while writing content to file.
Callback function : [optional] function that would be called once writing to the file is completed

Allowed encoding formats are
ascii
utf8
base64
Note : A new file with specified filename is created with data specified. If a file with the same name exists already, the content is overwritten. Care has to be taken as previous content of file could be lost.

Node FS File Operations

Example 1 – Write data to file in Node.js
nodejs-write-to-file-example.js

```
// include file system module

var fs = require('fs');

var data = "Hello World !"

// write data to file sample.html
fs.writeFile('sample.txt', data,
    // callback function that is called after writing file is done
    function(err) {
        if (err) throw err;
        // if no error
        console.log("Data is written to file successfully.")
});
```

Node FS File Operations

Write Content to File with Specified Encoding

We can also specify encoding to be followed by writeFile() method when it writes content to file.

nodejs-write-to-file-example-2.js

```
// include file system module

var fs = require('fs');

var data = "HELLO";

// write data to file sample.html
fs.writeFile('sample.txt',data, 'ascii',
    // callback function that is called after writing file is done
    function(err) {
        if (err) throw err;
        // if no error
        console.log("Data is written to file successfully.")
});
```

Node FS File Operations

Append to a File
To append data to file in Node.js, use Node FS appendFile() function for asynchronous file operation or Node FS appendFileSync() function for synchronous file operation.

Syntax – appendFile()
fs.appendFile(filepath, data, options, callback_function);
Callback function is mandatory and is called when appending data to file is completed.

Syntax – appendFileSync()

fs.appendFileSync(filepath, data, options);
where :

filepath [mandatory] is a String that specifies file path
data [mandatory] is what you append to the file
options [optional] to specify encoding/mode/flag
Note : If file specified does not exist, a new file is created with the name provided, and data is appended to the file.

Node FS File Operations

Example 1 – Node.js Append data to file asynchronously using appendFile()
To append data to a file asynchronously in Node.js, use appendFile() function of Node FS as shown below.

nodejs-append-to-file-example.js

```
// Example Node.js program to append data to file
var fs = require('fs');

var data = "\nLearn Node.js with the help of well built Node.js Tutorial.";

// append data to file
fs.appendFile('sample.txt',data, 'utf8',
    // callback function
    function(err) {
        if (err) throw err;
        // if no error
        console.log("Data is appended to file successfully.")
});
```

Node FS File Operations

Example 2 – Node.js Append data to file synchronously using appendFileSync()
To append data to a file synchronously in Node.js, use appendFileSync() function of Node FS as shown below.

nodejs-append-to-file-example-2.js

```
// Example Node.js program to append data to file
var fs = require('fs');

var data = "\nLearn Node.js with the help of well built Node.js Tutorial.";

// append data to file
fs.appendFileSync('sample.txt',data, 'utf8');
console.log("Data is appended to file successfully.")
```

Node.js Upload File To Server

Node.js Upload File –
Steps to Let User Upload File to Server in Node.js
To Upload File To Node.js Server, following is a step by step guide :

1. **Prerequisite modules**
We shall use http, fs and formidable modules for this example.

http : for server acitivities.
node fs : to save the uploaded file to a location at server.
formidable : to parse html form data.
If above mentioned modules are not installed already, you may install now using NPM. Run the following commands, in Terminal, to install the respective modules.

```
npm install http
npm install fs
npm install formidable
```

2. **Prepare a HTML Form**
Prepare a HTML page (upload_file.html) with the following form, which includes input tags for file upload and form submission.

```
<form action="fileupload" method="post" enctype="multipart/form-data">
    <input type="file" name="filetoupload">
    <input type="submit" value="Upload">
</form>
```

Node.js Upload File To Server

3. **Create a HTTP Server**

```
http.createServer(function (req, res) {
    if (req.url == '/uploadform') {
        // if request URL contains '/uploadform'
        // fill the response with the HTML file containing upload form
    } else if (req.url == '/fileupload') {
        // if request URL contains '/fileupload'
        // using formiddable module,
        // read the form data (which includes uploaded file)
        // and save the file to a location.
    }
}).listen(8086);
```

4. **File Saving**

Using formidable module, parse the form elements and save the file to a location. Once file is uploaded, you may respond with a message, saying file upload is successful. Initially, files are saved to a temporary location. We may use fs.rename() method, with the new path, to move the file to a desired location.

```
var form = new formidable.IncomingForm();
form.parse(req, function (err, fields, files) {
    // oldpath : temporary folder to which file is saved to
    var oldpath = files.filetoupload.path;
    var newpath = upload_path + files.filetoupload.name;
    // copy the file to a new location
    fs.rename(oldpath, newpath, function (err) {
        if (err) throw err;
        // you may respond with another html page
        res.write('File uploaded and moved!');
        res.end();
    });
});
```

Node.js Upload File To Server

upload_file.html

```
<!DOCTYPE html>
<html>
<head>
<title>Upload File</title>
<style>
    body{text-align:center;}
    form{display:block;border:1px solid black;padding:20px;}
</style>
</head>
<body>
    <h1>Upload files to Node.js Server</h1>

    <form action="fileupload" method="post" enctype="multipart/form-data">
        <input type="file" name="filetoupload">
        <input type="submit" value="Upload">
    </form>
</body>
</html
```

Node.js Upload File To Server

nodejs-upload-file.js

```
var http = require('http');
var fs = require('fs');
var formidable = require('formidable');
// html file containing upload form
var upload_html = fs.readFileSync("upload_file.html");
// replace this with the location to save uploaded files
var upload_path = "/home/arjun/workspace/nodejs/upload_file/";
http.createServer(function (req, res) {
    if (req.url == '/uploadform') {
      res.writeHead(200);
      res.write(upload_html);
      return res.end();
    } else if (req.url == '/fileupload') {
        var form = new formidable.IncomingForm();
        form.parse(req, function (err, fields, files) {
            // oldpath : temporary folder to which file is saved to
            var oldpath = files.filetoupload.path;
            var newpath = upload_path + files.filetoupload.name;
            // copy the file to a new location
            fs.rename(oldpath, newpath, function (err) {
                if (err) throw err;
                // you may respond with another html page
                res.write('File uploaded and moved!');
                res.end();
            });
        });
    }
}).listen(8086);
```

Node.js EventEmitter

Node.js allows us to create and handle custom events easily by using events module. Event module includes EventEmitter class which can be used to raise and handle custom events.

The following example demonstrates EventEmitter class for raising and handling a custom event.

Example: Raise and Handle Node.js events

```
// get the reference of EventEmitter class of events module
var events = require('events');

//create an object of EventEmitter class by using above reference
var em = new events.EventEmitter();

//Subscribe for FirstEvent
em.on('FirstEvent', function (data) {
    console.log('First subscriber: ' + data);
});

// Raising FirstEvent
em.emit('FirstEvent', 'This is my first Node.js event emitter example.');
```

In the above example, we first import the 'events' module and then create an object of EventEmitter class. We then specify event handler function using on() function. The on() method requires name of the event to handle and callback function which is called when an event is raised.

Node.js EventEmitter

The emit() function raises the specified event. First parameter is name of the event as a string and then arguments. An event can be emitted with zero or more arguments. You can specify any name for a custom event in the emit() function.

```
em.emit('FirstEvent', 'This is my first Node.js event emitter example.');
```

You can also use addListener() methods to subscribe for an event as shown below.

Example: EventEmitter

```
var emitter = require('events').EventEmitter;

var em = new emitter();

//Subscribe FirstEvent
em.addListener('FirstEvent', function (data) {
    console.log('First subscriber: ' + data);
});

//Subscribe SecondEvent
em.on('SecondEvent', function (data) {
    console.log('First subscriber: ' + data);
});

// Raising FirstEvent
em.emit('FirstEvent', 'This is my first Node.js event emitter example.');

// Raising SecondEvent
em.emit('SecondEvent', 'This is my second Node.js event emitter example.');
```

Node.js EventEmitter

The following table lists all the important methods of EventEmitter class.

EventEmitter Methods	Description
emitter.addListener(event, listener)	Adds a listener to the end of the listeners array for the specified event. No checks are made to see if the listener has already been added.
emitter.on(event, listener)	Adds a listener to the end of the listeners array for the specified event. No checks are made to see if the listener has already been added. It can also be called as an alias of emitter.addListener()
emitter.once(event, listener)	Adds a one time listener for the event. This listener is invoked only the next time the event is fired, after which it is removed.
emitter.removeListener(event, listener)	Removes a listener from the listener array for the specified event. Caution: changes array indices in the listener array behind the listener.
emitter.removeAllListeners([event])	Removes all listeners, or those of the specified event.
emitter.setMaxListeners(n)	By default EventEmitters will print a warning if more than 10 listeners are added for a particular event.
emitter.getMaxListeners()	Returns the current maximum listener value for the emitter which is either set by emitter.setMaxListeners(n) or defaults to EventEmitter.defaultMaxListeners.
emitter.listeners(event)	Returns a copy of the array of listeners for the specified event.
emitter.emit(event[, arg1][, arg2][, ...])	Raise the specified events with the supplied arguments.
emitter.listenerCount(type)	Returns the number of listeners listening to the type of event.

Node.js EventEmitter

Common Patterns for EventEmitters

There are two common patterns that can be used to raise and bind an event using EventEmitter class in Node.js.

- Return EventEmitter from a function
- Extend the EventEmitter class

Node.js EventEmitter

Return EventEmitter from a function

In this pattern, a constructor function returns an EventEmitter object, which was used to emit events inside a function. This EventEmitter object can be used to subscribe for the events:

Example: Return EventEmitter from a function

```
var emitter = require('events').EventEmitter;
function LoopProcessor(num) {
    var e = new emitter();
    setTimeout(function () {
        for (var i = 1; i <= num; i++) {
            e.emit('BeforeProcess', i);
            console.log('Processing number:' + i);
             e.emit('AfterProcess', i);
        }
    }
    , 2000)
    return e;
}
var lp = LoopProcessor(3);

lp.on('BeforeProcess', function (data) {
    console.log('About to start the process for ' + data);
});

lp.on('AfterProcess', function (data) {
    console.log('Completed processing ' + data);
});
```

Node.js EventEmitter

Output:

```
About to start the process for 1
Processing number:1
Completed processing 1
About to start the process for 2
Processing number:2
Completed processing 2
About to start the process for 3
Processing number:3
Completed processing 3
```

In the above LoopProcessor() function, first we create an object of EventEmitter class and then use it to emit 'BeforeProcess' and 'AfterProcess' events. Finally, we return an object of EventEmitter from the function. So now, we can use the return value of LoopProcessor function to bind these events using on() or addListener() function.

Node.js EventEmitter

Extend EventEmitter Class:

In this pattern, we can extend the constructor function from EventEmitter class to emit the events.

Example: Extend EventEmitter Class

```
var emitter = require('events').EventEmitter;
var util = require('util');
function LoopProcessor(num) {
    var me = this;
    setTimeout(function () {
for (var i = 1; i <= num; i++) {
            me.emit('BeforeProcess', i);
console.log('Processing number:' + i);
me.emit('AfterProcess', i);
        }
    }
    , 2000)
return this;
}
util.inherits(LoopProcessor, emitter)
var lp = new LoopProcessor(3);
lp.on('BeforeProcess', function (data) {
    console.log('About to start the process for ' + data);
});
lp.on('AfterProcess', function (data) {
    console.log('Completed processing ' + data);
});
```

Node.js EventEmitter

Output:

```
About to start the process for 1
Processing number:1
Completed processing 1
About to start the process for 2
Processing number:2
Completed processing 2
About to start the process for 3
Processing number:3
Completed processing 3
```

In the above example, we have extended LoopProcessor constructor function with EventEmitter class using util.inherits() method of utility module. So, you can use EventEmitter's methods with LoopProcessor object to handle its own events.

In this way, you can use EventEmitter class to raise and handle custom events in Node.js.

Frameworks for Node.js

There are various third party open-source frameworks available in Node Package Manager which makes Node.js application development faster and easy. You can choose an appropriate framework as per your application requirements.

The following table lists frameworks for Node.js:

Open-Source Framework	Description
Express.js	Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. This is the most popular framework as of now for Node.js.
Geddy	Geddy is a simple, structured web application framework for Node.js based on MVC architecture.
Locomotive	Locomotive is MVC web application framework for Node.js. It supports MVC patterns, RESTful routes, and convention over configuration, while integrating seamlessly with any database and template engine. Locomotive builds on Express, preserving the power and simplicity you've come to expect from Node.
Koa	Koa is a new web framework designed by the team behind Express, which aims to be a smaller, more expressive, and more robust foundation for web applications and APIs.
Total.js	Totaljs is free web application framework for building web sites and web applications using JavaScript, HTML and CSS on Node.js
Hapi.js	Hapi is a rich Node.js framework for building applications and services.
Keystone	Keystone is the open source framework for developing database-driven websites, applications and APIs in Node.js. Built on Express and MongoDB.
Derbyjs	Derby support single-page apps that have a full MVC structure, including a model provided by Racer, a template and styles based view, and controller code with application logic and routes.
Sails.js	Sails makes it easy to build custom, enterprise-grade Node.js apps. It is designed to emulate the familiar MVC pattern of frameworks like Ruby on Rails, but with support for the requirements of modern apps: data-driven APIs with a scalable, service-oriented architecture. It's especially good for building chat, realtime dashboards, or multiplayer games; but you can use it for any web application project - top to bottom.
Meteor	Meteor is a complete open source platform for building web and mobile apps in pure JavaScript.

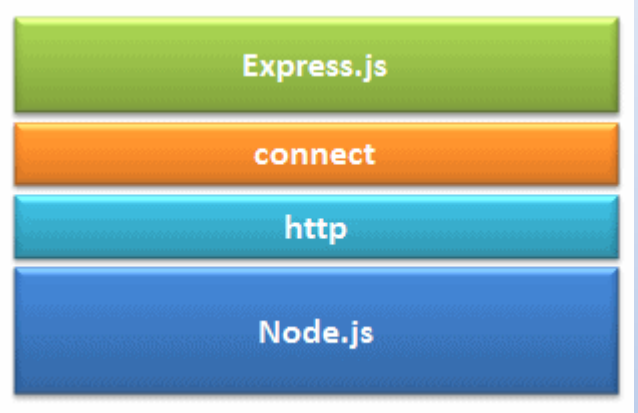
Express Js

"Express is a fast, unopinionated minimalist web framework for Node.js" - official web site: **Expressjs.com**

Express.js is a web application framework for Node.js. It provides various features that make web application development fast and easy which otherwise takes more time using only Node.js.

Express.js is based on the Node.js middleware module called connect which in turn uses http module.

So, any middleware which is based on connect will also work with Express.js.



Advantages of Express.js

1. Makes Node.js web application development fast and easy.
2. Easy to configure and customize.
3. Allows you to define routes of your application based on HTTP methods and URLs.
4. Includes various middleware modules which you can use to perform additional tasks on request and response.
5. Easy to integrate with different template engines like Jade, Vash, EJS etc.
6. Allows you to define an error handling middleware.
7. Easy to serve static files and resources of your application.
8. Allows you to create REST API server.
9. Easy to connect with databases such as MongoDB, Redis, MySQL

Express.js Web Application

Express.js provides an easy way to create web server and render HTML pages for different HTTP requests by configuring routes for your application.

Web Server

First of all, import the Express.js module and create the web server as shown below.

app.js: Express.js Web Server

```
var express = require('express');
var app = express();
// define routes here..
var server = app.listen(5000, function () {
    console.log('Node server is running..');
});
```

In the above example, we imported Express.js module using require() function. The express module returns a function. This function returns an object which can be used to configure Express application (app in the above example).

The app object includes methods for routing HTTP requests, configuring middleware, rendering HTML views and registering a template engine.

The app.listen() function creates the Node.js web server at the specified host and port. It is identical to Node's http.Server.listen() method.

Run the above example using node app.js command and point your browser to http://localhost:5000. It will display Cannot GET / because we have not configured any routes yet.

Express.js Web Application

Configure Routes

Use app object to define different routes of your application. The app object includes get(), post(), put() and delete() methods to define routes for HTTP GET, POST, PUT and DELETE requests respectively.

The following example demonstrates configuring routes for HTTP requests.

Example: Configure Routes in Express.js

```
var express = require('express');
var app = express();
app.get('/', function (req, res) {
    res.send('<html><body><h1>Hello World</h1></body></html>');
});
app.post('/submit-data', function (req, res) {
    res.send('POST Request');
});
app.put('/update-data', function (req, res) {
    res.send('PUT Request');
});
app.delete('/delete-data', function (req, res) {
    res.send('DELETE Request');
});
var server = app.listen(5000, function () {
    console.log('Node server is running..');
});
```

In the above example, app.get(), app.post(), app.put() and app.delete() methods define routes for HTTP GET, POST, PUT, DELETE respectively. The first parameter is a path of a route which will start after base URL. The callback function includes request and response object which will be executed on each request.

Express.js Web Application

Handle POST Request

Use app object to define different routes of your application. The app object includes get(), post(), put() and delete() methods to define routes for HTTP GET, POST, PUT and DELETE requests respectively.

The following example demonstrates configuring routes for HTTP requests.

Example: Configure Routes in Express.js

```
var express = require('express');
var app = express();
app.get('/', function (req, res) {
    res.send('<html><body><h1>Hello World</h1></body></html>');
});
app.post('/submit-data', function (req, res) {
    res.send('POST Request');
});
app.put('/update-data', function (req, res) {
    res.send('PUT Request');
});
app.delete('/delete-data', function (req, res) {
    res.send('DELETE Request');
});
var server = app.listen(5000, function () {
    console.log('Node server is running..');
});
```

In the above example, app.get(), app.post(), app.put() and app.delete() methods define routes for HTTP GET, POST, PUT, DELETE respectively. The first parameter is a path of a route which will start after base URL. The callback function includes request and response object which will be executed on each request.

Express.js Middleware

What is Middleware?

Middleware is a function that can access request and response objects and can also use next function in the application's request-response cycle.

In this tutorial, we will learn how to define a middleware function in Node.js Express application and how to make a call to the middleware function.

Middleware Terminology

request – is the HTTP request that reaches the Express application when a client makes HTTP request like PUT, GET, etc. It contains properties like query string, url parameters, headers, etc.

response – object represents the HTTP response that an Express application sends when it gets an HTTP request.

request-response cycle – The cycle of operations that get executed starting with a request hitting the Express application till a response leaves the application for the request.

middleware stack – stack of middleware functions that get executed for a request-response cycle.

Define Middleware Function

As we have already mentioned in the definition of middleware function, it has access to request, response objects and next function.

The syntax is same as that of a JavaScript Function. It accepts request, response objects and next function as arguments.

```
function logger(req, res, next) {

 }
```

here, logger is the function name, req is the HTTP request object, res is the Node Response Object and next is the next function in request-response cycle.

Express.js Middleware

You can access all the properties and methods of request object req.

Similarly, you can access all the properties and methods of response object res.

Calling next() function inside the middleware function is optional. If you use next() statement, the execution continues with the next middleware function in request-response cycle. If you do not call next() function, the execution for the given request stops here.

```
function logger(req, res, next) {
    // your code
    next()  // calls the next function in the middleware stack
 }
```

Call Middleware

In an Express application, you call middleware using use function on application object.

```
var express = require('express')
var app = express()
function logger(req, res, next) {
   // your code
   next()
}
app.use(logger)
```

Express.js Middleware

Express.js Middleware Example

In this example, we will define a middleware called logger which logs the current time and query string to the console.

```
app.js
var express = require('express')
var app = express()
// define middleware function
function logger(req, res, next) {
   console.log(new Date(), req.url)
   next()
}
// calls logger:middleware for each request-response cycle
app.use(logger)
// route that gets executed for the path '/'
app.get('/', function (req, res) {
   res.send('This is a basic Example of logger')
})
// start the server
var server = app.listen(8000, function(){
    console.log('Listening on port 8000...')
})
```

Start this application and hit the following urls in your browser.

http://localhost:8000/

http://localhost:8000/hello-page/

REST API

REST is acronym for Representational State Transfer. It is architectural style for distributed hypermedia systems and was first presented by Roy Fielding in 2000 in his famous dissertation. Like any other architectural style, REST also does have it's own 6 guiding constraints which must be satisfied if an interface needs to be referred as **RESTful**. These principles are listed below:

REST API

Guiding Principles of REST:

1. **Client–server** – By separating the user interface concerns from the data storage concerns, we improve the portability of the user interface across multiple platforms and improve scalability by simplifying the server components.
2. **Stateless** – Each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server. Session state is therefore kept entirely on the client.
3. **Cacheable** – Cache constraints require that the data within a response to a request be implicitly or explicitly labeled as cacheable or non-cacheable. If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests.
4. **Uniform interface** – By applying the software engineering principle of generality to the component interface, the overall system architecture is simplified and the visibility of interactions is improved. In order to obtain a uniform interface, multiple architectural constraints are needed to guide the behavior of components. REST is defined by four interface constraints: identification of resources; manipulation of resources through representations; self-descriptive messages; and, hypermedia as the engine of application state.
5. **Layered system** – The layered system style allows an architecture to be composed of hierarchical layers by constraining component behavior such that each component cannot "see" beyond the immediate layer with which they are interacting.
6. **Code on demand (optional)** – REST allows client functionality to be extended by downloading and executing code in the form of applets or scripts. This simplifies clients by reducing the number of features required to be pre-implemented

REST API

Resource:

The key abstraction of information in REST is a resource. Any information that can be named can be a resource: a document or image, a temporal service, a collection of other resources, a non-virtual object (e.g. a person), and so on. REST uses a resource identifier to identify the particular resource involved in an interaction between components.

The state of the resource at any particular timestamp is known as resource representation. A representation consists of data, metadata describing the data and hypermedia links which can help the clients in transition to the next desired state.

REST API

REST and HTTP are not same !!

In the REST architectural style, data and functionality are considered resources and are accessed using Uniform Resource Identifiers (URIs). The resources are acted upon by using a set of simple, well-defined operations. The clients and servers exchange representations of resources by using a standardized interface and protocol – typically HTTP.

REST API

Resource Methods:

Another important thing associated with REST is resource methods to be used to perform the desired transition. A large number of people wrongly relate resource methods to HTTP GET/PUT/POST/DELETE methods.

Roy Fielding has never mentioned any recommendation around which method to be used in which condition. All he emphasizes is that it should be uniform interface. If you decide HTTP POST will be used for updating a resource – rather than most people recommend HTTP PUT – it's alright and application interface will be RESTful.

REST API

REST Resource Naming Guide

- A resource can be a singleton or a collection - /customers
- A resource may contain sub-collection resources, example

  /customers/{customerId}/accounts/{accountId}

REST APIs use Uniform Resource Identifiers (URIs) to address resources. REST API designers should create URIs that convey a REST API's resource model to its potential client developers. When resources are named well, an API is intuitive and easy to use.

REST API

REST Resource Naming Best Practices

- Use nouns to represent resources

  http://api.example.com/device-management/managed-devices

- Consistency is the key

  http://api.example.com/device-management/managed-devices/{id}/scripts/{id}

- Never use CRUD function names in URIs

  HTTP POST http://api.example.com/device-management/managed-devices //Create new Device

- Use query component to filter URI collection

  http://api.example.com/device-management/managed-devices?region=USA&brand=XYZ

REST API

Http Methods:

- HTTP GET - to retrieve resource representation/information only
- HTTP POST - to create new subordinate resources
- HTTP PUT - to update existing resource
- HTTP DELETE - to delete resources
- HTTP PATCH - to make partial update on a resource
- Note: HTTP Methods – REST API Verbs (restfulapi.net)

REST API

HTTP METHOD	CRUD	ENTIRE COLLECTION (E.G. /USERS)	SPECIFIC ITEM (E.G. /USERS/123)
POST	Create	201 (Created), ‘Location’ header with link to /users/{id} containing new ID.	Avoid using POST on single resource
GET	Read	200 (OK), list of users. Use pagination, sorting and filtering to navigate big lists.	200 (OK), single user. 404 (Not Found), if ID not found or invalid.
PUT	Update/Replace	405 (Method not allowed), unless you want to update every resource in the entire collection of resource.	200 (OK) or 204 (No Content). Use 404 (Not Found), if ID not found or invalid.
PATCH	Partial Update/Modify	405 (Method not allowed), unless you want to modify the collection itself.	200 (OK) or 204 (No Content). Use 404 (Not Found), if ID not found or invalid.
DELETE	Delete	405 (Method not allowed), unless you want to delete the whole collection — use with caution.	200 (OK). 404 (Not Found), if ID not found or invalid.

REST API

Http Status Codes:

- HTTP defines these standard status codes that can be used to convey the results of a client’s request. The status codes are divided into the five categories.
- 1xx: Informational – Communicates transfer protocol-level information.
- 2xx: Success – Indicates that the client’s request was accepted successfully.
- 3xx: Redirection – Indicates that the client must take some additional action in order to complete their request.
- 4xx: Client Error – This category of error status codes points the finger at clients.
- 5xx: Server Error – The server takes responsibility for these error status codes.

Data Access in Node.js

Data Access

Node.js supports all kinds of databases no matter if it is a relational database or NoSQL database. However, NoSQL databases like MongoDb are the best fit with Node.js.

To access the database from Node.js, you first need to install drivers for the database you want to use.

The following table lists important relational databases and respective drivers.

Note: The above database list is not limited. There are many other databases and drivers available to be used with Node.js. Also, there are many drivers available for each database. So, choose a driver carefully based on your need

Relational Databases	Driver	NPM Command
MS SQL Server	mssql	npm install mssql
Oracle	oracledb	npm install oracledb
MySQL	MySQL	npm install mysql
PostgreSQL	pg	npm install pg
SQLite	node-sqlite3	npm install node-sqlite

NoSQL Databases	Driver	NPM Command
MongoDB	mongodb	npm install mongodb
Cassandra	cassandra-driver	npm install cassandra-driver
LevelDB	leveldb	npm install level levelup leveldown
RavenDB	ravendb	npm install ravendb
Neo4j	neo4j	npm install neo4j
Redis	redis	npm install redis
CouchDB	nano	npm install nano

Node.js MySQL

Node.js MySQL is one of the external libraries of Node.js. It helps Node.js developers to connect to MySQL database and execute MySQL Queries.

Install MySQL in Node.js

As Node.js MySQL is an external module, it can be installed using NPM (Node Package Manager).

```
$ npm install mysql
```

After successful install, you may use MySQL module in node.js programs by declaring its usage with a require statement as shown below.

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

Note – If MySQL module is not installed, but used in Node.js programs, you might get the Error : Cannot find module 'mysql'.

Create Connection to MySQL Database

To create a connection variable with IP Address (of server where MySQL server is running), User Name and Password (of user that has access the MySQL database). An example is provided below :

```
var con = mysql.createConnection({
  host: "localhost",    // ip address of server running mysql
  user: "arjun",    // user name to your mysql database
  password: "pastudentsDBssword",    // corresponding password
  database: ""  // use this database to querying context
});
```

Node.js MySQL

SELECT FROM Query

MySQL SELECT Query is used to select some of the records (with some of their properties if required) of a table.

```
con.query("SELECT * FROM studentsDB.students", function (err, result, fields) {
    // if any error while executing above query, throw error
    if (err) throw err;
    // if there is no error, you have the result
    console.log(result);
});
```

Node.js MySQL

Example:

```
var mysql = require('mysql');
// create a connection variable with the required details
var con = mysql.createConnection({
  host: "localhost",    // ip address of server running mysql
  user: "amit",    // user name to your mysql database
  password: "password", // corresponding password
  database: "studentsDB" // use the specified database
});
// make to connection to the database.
con.connect(function(err) {
  if (err) throw err;
  // if connection is successful
  con.query("SELECT * FROM students", function (err, result, fields) {
    // if any error while executing above query, throw error
    if (err) throw err;
    // if there is no error, you have the result
    console.log(result);
  });
});
```

Node.js Mongodb

The prerequisite to connect to a MongoDB, is having MongoDB installed in the computer:

Steps to Connect to MongoDB via Node.js

Mongodb compass url for localhost: mongodb://localhost:27017/?readPreference=primary&appname=MongoDB%20Compass&directConnection=true&ssl=false

To connect to MongoDB from Node.js Application, following is a step by step guide

Step 1: Start MongoDB service ((not required if compass is installed)

Run the following command to start MongoDB Service.

sudo service mongod start

Step 2: Install mongodb package using npm (if not installed already).

npm install mongodb

Step 3: Prepare the url.

A simple hack to know the base url of MongoDB Service is to Open a Terminal and run Mongo Shell.

arjun@nodejs:~$ mongo

mongodb://127.0.0.1:27017

Step 4: With the help of mongodb package, create a MongoClient and connect to the url.

Node.js Mongodb

Example:

```
// URL at which MongoDB service is running
var url = "mongodb://localhost:27017";

// A Client to MongoDB
var MongoClient = require('mongodb').MongoClient;

// Make a connection to MongoDB Service
MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  console.log("Connected to MongoDB!");
  db.close();
});
```

Node.js Mongodb

Create Database:

```
// newdb is the new database we create
var url = "mongodb://localhost:27017/newdb";

// create a client to mongodb
var MongoClient = require('mongodb').MongoClient;

// make client connect to mongo service
MongoClient.connect(url, function(err, db) {
    if (err) throw err;
    console.log("Database created!");
    // print database name
    console.log("db object points to the database : "+ db.databaseName);
    // after completing all the operations with db, close it.
    db.close();
});
```

Node.js Mongodb

Create Collection:

```
// we create 'users' collection in newdb database
var url = "mongodb://localhost:27017/newdb";

// create a client to mongodb
var MongoClient = require('mongodb').MongoClient;

// make client connect to mongo service
MongoClient.connect(url, function(err, db) {
    if (err) throw err;
    // db pointing to newdb
      var dbo = db.db(" newdb ");
    console.log("Switched to "+db.databaseName+" database");
    // create 'users' collection in newdb database
    db.createCollection("users", function(err, result) {
        if (err) throw err;
        console.log("Collection is created!");
        // close the connection to db when you are done with it
        db.close();
    });
});
```

Node.js Mongodb

Insert Document:

```
// we create 'users' collection in newdb database
var url = "mongodb://localhost:27017/newdb";
// create a client to mongodb
var MongoClient = require('mongodb').MongoClient;
// make client connect to mongo service
MongoClient.connect(url, function(err, db) {
    if (err) throw err;
    // db pointing to newdb
     var dbo = db.db(" newdb ");
    console.log("Switched to "+db.databaseName+" database");
    // document to be inserted
    var doc = { name: "Roshan", age: "22" };
    // insert document to 'users' collection using insertOne
    db.collection("users").insertOne(doc, function(err, res) {
        if (err) throw err;
        console.log("Document inserted");
        // close the connection to db when you are done with it
        db.close();
    });
});
```

Node.js Mongoose

Node.js **Mongoose** is a MongoDB object modeling tool designed to work in an asynchronous environment.

With the help of Mongoose, we can model our data.

For example consider that we are operating a store. Store has items. Each item could have properties : name, id, price, discount price, etc. With Mongoose we can model our items and do insertions or reads from the MongoDB Collection in terms of model objects, not bothering about the details of an object. Mongoose provides abstraction at Model level.

To install Mongoose package, use npm (Node Package Manager)

```
npm install mongoose
```

Using Mongoose:

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

Connect to MongoDB:

```
var mongoose = require('mongoose');
mongoose.connect('mongodb://localhost:27017/database_name');
```

Node.js Mongoose

Example:

```
var mongoose = require('mongoose');
mongoose.connect('mongodb://localhost:27017/tutorialkart');

var db = mongoose.connection;

db.on('error', console.error.bind(console, 'connection error:'));

db.once('open', function() {
  console.log("Connection Successful!");
});
```

Node.js Mongoose

Mongoose – Define a Model

To define a model, derive a custom schema from Mongoose's Schema and compile the schema to a model.

How to recognize a Model

Let us consider that we are operating a bookstore and we need to develop a Node.js Application for maintaining the book store. Also we chose MongoDB as the database for storing data regarding books. The simplest item of transaction here is a book. Hence, we shall define a model called Book and transact objects of Book Model between Node.js and MongoDB. Mongoose helps us to abstract at Book level, during transactions with the database.

Derive a custom schema

Following is an example where we derive a custom Schema from Mongoose's Schema.

```
var BookSchema = mongoose.Schema({
  name: String,
  price: Number,
  quantity: Number
});
```

Node.js Mongoose

Compile Schema to Model

Once we derive a custom schema, we could compile it to a model.

Following is an example where we define a model named Book with the help of BookSchema.

```
var Book = mongoose.model('Book', BookSchema, <collection_name>);
```

<collection_name> is the name of the collection you want the documents go to.

Initialize a Document

We may now use the model to initialize documents of the Model.

```
var book1 = new Book({ name: 'Introduction to Mongoose', price: 10, quantity: 25 });
```

book1 is a Document of model Book.

Node.js Mongoose

Mongoose – Insert Document to MongoDB

Insert Document to MongoDB – To insert a single document to MongoDB, call save() method on document instance. Callback function(err, document) is an optional argument to save() method. Insertion happens asynchronously and any operations dependent on the inserted document has to happen in callback function for correctness.

```
var mongoose = require('mongoose');
// make a connection
mongoose.connect('mongodb://localhost:27017/tutorialkart');
// get reference to database
var db = mongoose.connection;
db.on('error', console.error.bind(console, 'connection error:'));
db.once('open', function() {
    console.log("Connection Successful!");
    // define Schema
    var BookSchema = mongoose.Schema({
      name: String,
      price: Number,
      quantity: Number
    });
    // compile schema to model
    var Book = mongoose.model('Book', BookSchema, 'bookstore');
    // a document instance
    var book1 = new Book({ name: 'Introduction to Mongoose', price: 10, quantity: 25 });
    // save model to database
    book1.save(function (err, book) {
      if (err) return console.error(err);
      console.log(book.name + " saved to bookstore collection.");
    });
});
```

Node.js Mongoose

Mongoose – Insert Multiple Documents to MongoDB

To insert Multiple Documents to MongoDB using Mongoose, use Model.collection.insert(docs_array, options, callback_function); method. Callback function has error and inserted_documents as arguments.

Syntax of insert() method

Model.collection.insert(docs, options, callback)

where

docs is the array of documents to be inserted;

options is an optional configuration object – see the docs

callback(err, docs) will be called after all documents get saved or an error occurs. On success, docs is the array of persisted documents.

Example – Insert Multiple Documents to MongoDB via Node.js

In this next example, we will write a Node.js script that inserts Multiple Documents to MongoDB Collection 'bookstore' using Mongoose module.

Node.js Mongoose

```
var mongoose = require('mongoose');
// make a connection
mongoose.connect('mongodb://localhost:27017/tutorialkart');
// get reference to database
var db = mongoose.connection;
db.on('error', console.error.bind(console, 'connection error:'));
db.once('open', function() {
    console.log("Connection Successful!");
    // define Schema
    var BookSchema = mongoose.Schema({
      name: String,
      price: Number,
      quantity: Number
    });
    // compile schema to model
    var Book = mongoose.model('Book', BookSchema, 'bookstore');
    // documents array
    var books = [{ name: 'Mongoose Tutorial', price: 10, quantity: 25 },
                 { name: 'NodeJS tutorial', price: 15, quantity: 5 },
                 { name: 'MongoDB Tutorial', price: 20, quantity: 2 }];
    // save multiple documents to the collection referenced by Book Model
    Book.collection.insert(books, function (err, docs) {
      if (err){
          return console.error(err);
      } else {
        console.log("Multiple documents inserted to Collection");
      }
    });
});
```

References:

Reference:

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