Node JS, Express JS MongoDB

NAVEEN SAGGAM
http://github.com/thenaveensaggam

What is Node JS

Node.js is an open-source, cross-platform runtime environment used for development of server-side web applications. Node.js applications are written in JavaScript and can be run on a wide variety of operating systems.

Node.js is based on an event-driven architecture and a non-blocking Input/Output API that is designed to optimize an application's throughput and scalability for real-time web applications.

Over a long period of time, the framework available for web development were all based on a stateless model. A stateless model is where the data generated in one session (such as information about user settings and events that occurred) is not maintained for usage in the next session with that user.

A lot of work had to be done to maintain the session information between requests for a user. But with Node.js there is finally a way for web applications to have a real-time, two-way connections, where both the client and server can initiate communication, allowing them to exchange data freely.

Why use Node.js?

We will have a look into the real worth of Node.js in the coming chapters, but what is it that makes this framework so famous. Over the years, most of the applications were based on a stateless request-response framework. In these sort of applications, it is up to the developer to ensure the right code was put in place to ensure the state of web session was maintained while the user was working with the system.

But with Node.js web applications, you can now work in real-time and have a 2-way communication. The state is maintained, and the either the client or server can start the communication.

Features of Node.js

Let's look at some of the key features of Node.js

Asynchronous event driven IO helps concurrent request handling – This is probably the biggest selling points of Node.js. This feature basically means that if a request is received by Node for some Input/Output operation, it will execute the operation in the background and continue with processing other requests.

This is quite different from other programming languages. A simple example of this is given in the code below

```
var fs = require('fs');
        fs.readFile("Sample.txt",function(error,data)
        {
             console.log("Reading Data completed");
    });
```

The above code snippet looks at reading a file called Sample.txt. In other programming languages, the next line of processing would only happen once the entire file is read.

But in the case of Node.js the important fraction of code to notice is the declaration of the function ('function(error,data)'). This is known as a callback function.

So what happens here is that the file reading operation will start in the background. And other processing can happen simultaneously while the file is being read. Once the file read operation is completed, this anonymous function will be called and the text "Reading Data completed" will be written to the console log.

Node uses the V8 JavaScript Runtime engine, the one which is used by Google Chrome. Node has a wrapper over the JavaScript engine which makes the runtime engine much faster and hence processing of requests within Node also become faster.

Handling of concurrent requests – Another key functionality of Node is the ability to handle concurrent connections with a very minimal overhead on a single process.

The Node.js library used JavaScript – This is another important aspect of development in Node.js. A major part of the development community are already well versed in javascript, and hence, development in Node.js becomes easier for a developer who knows javascript.

There are an Active and vibrant community for the Node.js framework. Because of the active community, there are always keys updates made available to the framework. This helps to keep the framework always up-to-date with the latest trends in web development.

Who uses Node.js

Node.js is used by a variety of large companies. Below is a list of a few of them.

1) Paypal – A lot of sites within Paypal have also started the transition onto Node.js.
2) LinkedIn - LinkedIn is using Node.js to power their Mobile Servers, which powers the iPhone, Android, and Mobile Web products.
3) Mozilla has implemented Node.js to support browser APIs which has half a billion installs.
4) Ebay hosts their HTTP API service in Node.js

When to Use Node.js

Node.js is best for usage in streaming or event-based real-time applications like

Chat applications

Game servers – Fast and high-performance servers that need to processes thousands of requests at a time, then this is an ideal framework.

Good for collaborative environment – This is good for environments which manage document. In document management environment you will have multiple people who post their documents and do constant changes by checking out and checking in documents. So Node.js is good for these environments because the event loop in Node.js can be triggered whenever documents are changed in a document managed environment.

Advertisement servers – Again here you could have thousands of request to pull advertisements from the central server and Node.js can be an ideal framework to handle this.

Streaming servers – Another ideal scenario to use Node is for multimedia streaming servers wherein clients have request's to pull different multimedia contents from this server.

Node.js is good when you need high levels of concurrency but less amount of dedicated CPU time.

Best of all, since Node.js is built on javascript, it's best suited when you build client-side applications which are based on the same javascript framework.

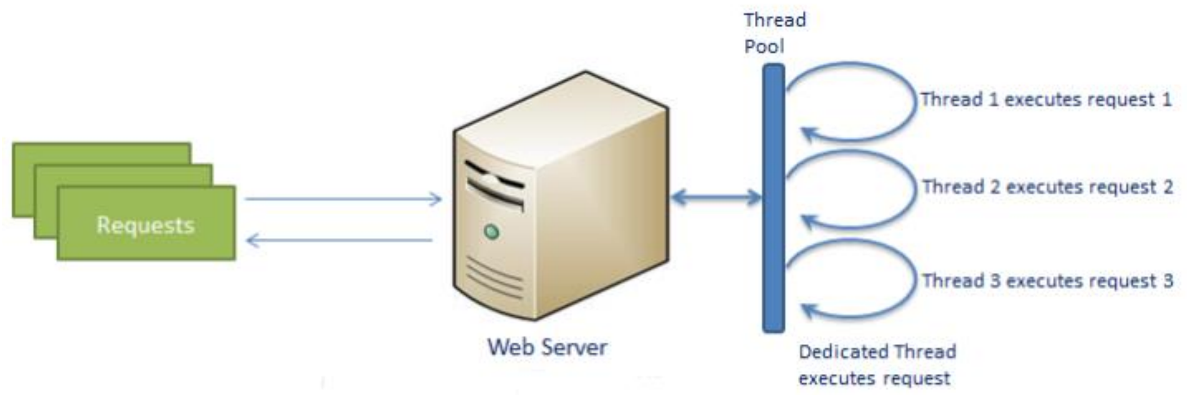
When to not use Node.js

Node.js can be used for a lot of applications with various purpose, the only scenario where it should not be used is if there are long processing times which is required by the application.

Node is structured to be single threaded. If any application is required to carry out some long running calculations in the background. So if the server is doing some calculation, it won't be able to process any other requests. As discussed above, Node.js is best when processing needs less dedicated CPU time.

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.

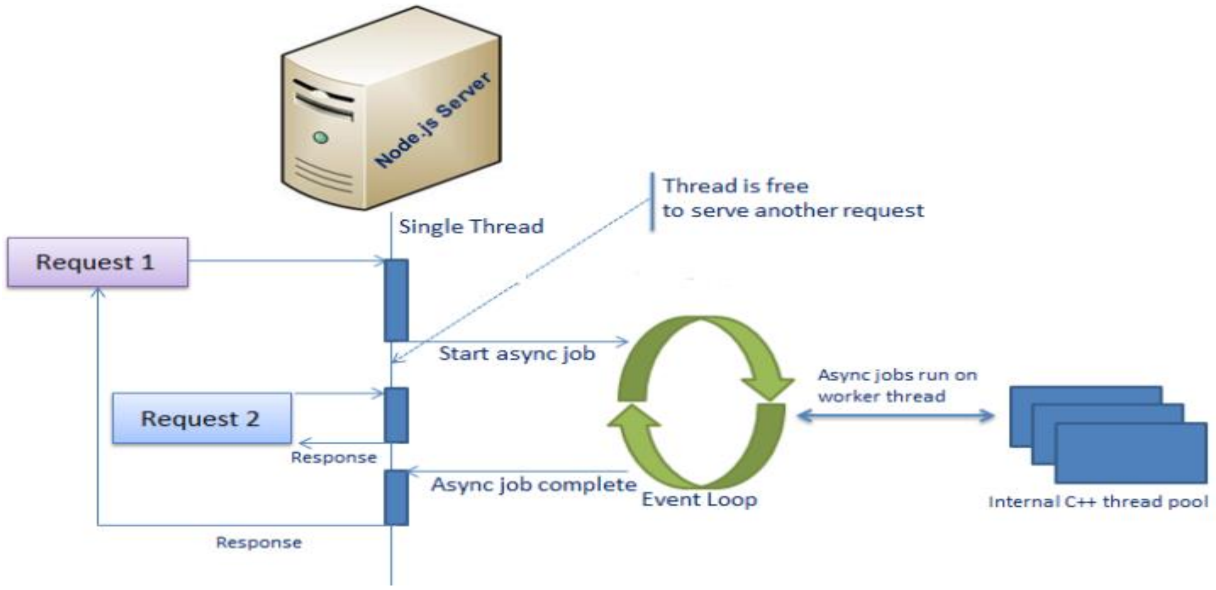

Thread Pool
Requests
Web Server
Thread 1 executes request 1
Thread 2 executes request 2
Thread 3 executes request 3
Dedicated Thread executes request

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 libevfor the event loop which in turn uses internal C++ thread pool to provide asynchronous I/O.

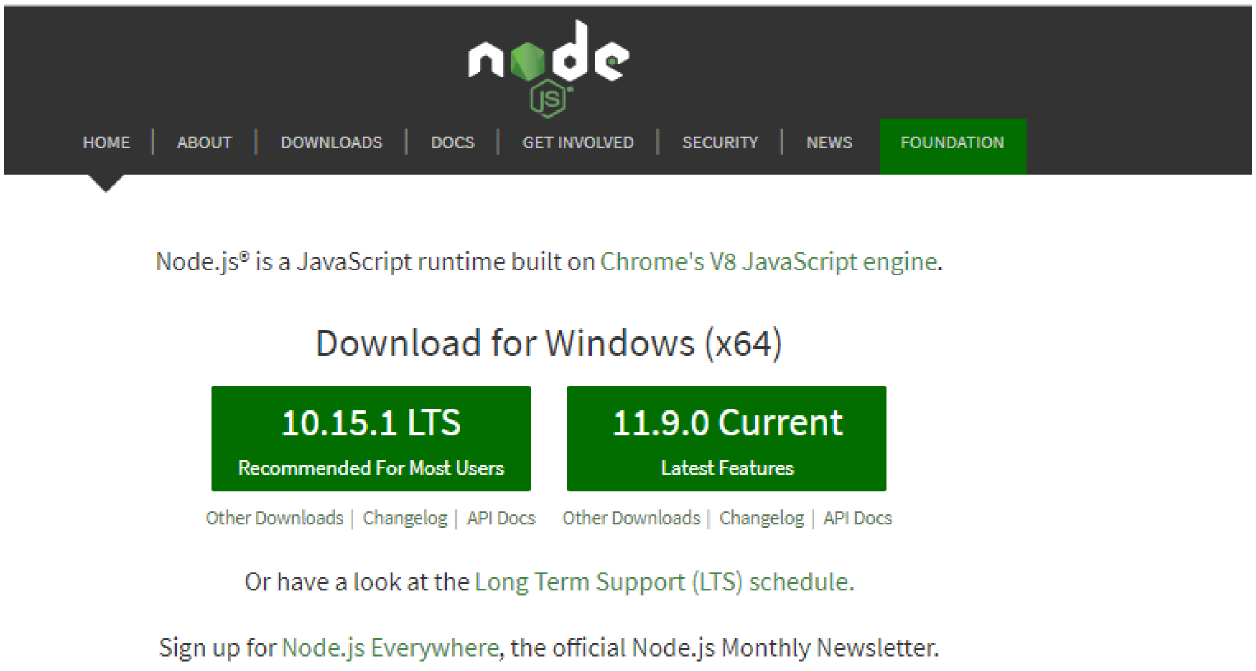
The following figure illustrates asynchronous web server model using Node.js.



Node.js process model increases the performance and scalability with a few caveats. Node.js is not fit for an application which performs CPU-intensive operations like image processing or other heavy computation work because it takes time to process a request and thereby blocks the single thread.

Node JS Installation

We can install the node JS from the official website https://nodejs.org/en/



Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.

Download for Windows (x64)

10.15.1 LTS
Recommended For Most Users

11.9.0 Current
Latest Features

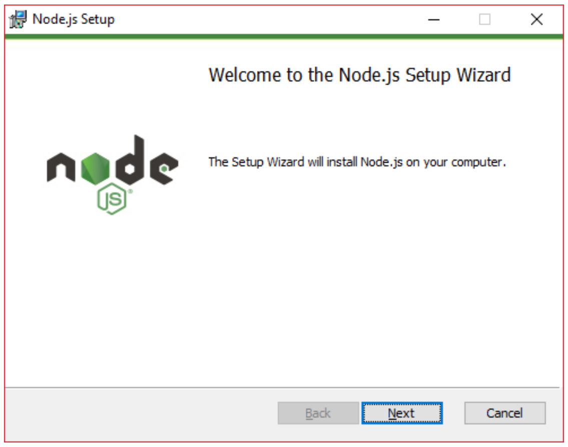
Other Downloads | Changelog | API Docs   Other Downloads | Changelog | API Docs

Or have a look at the Long Term Support (LTS) schedule.

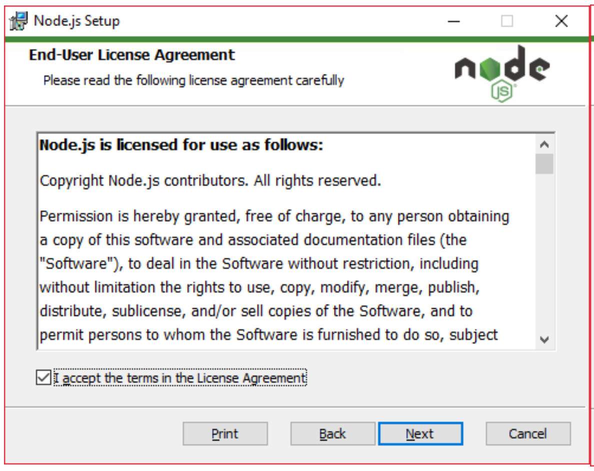
Sign up for Node.js Everywhere, the official Node.js Monthly Newsletter.

Choose any installable version, it downloads a '.msi' file in your system. For Mac Users click on other downloads option below each version.

Now install the downloaded 'node.10.15.1.msi' file.


Node.js Setup
Welcome to the Node.js Setup Wizard
The Setup Wizard will install Node.js on your computer.
Back
Next
Cancel

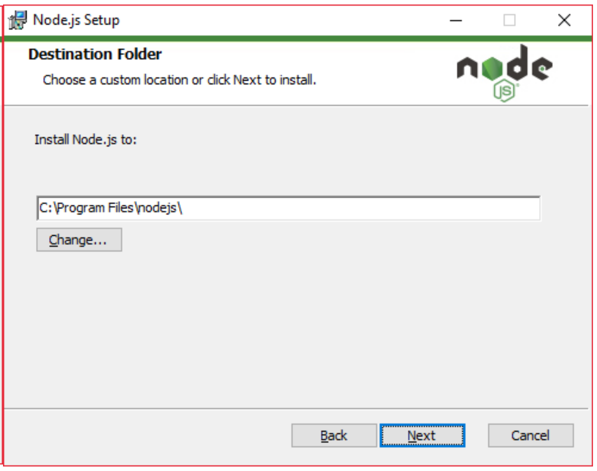
Click on Next Button , and accept the licence.


Node.js Setup
End-User License Agreement
Please read the following license agreement carefully
I accept the terms in the License Agreement
Print
Back
Next
Cancel

Node.js is licensed for use as follows:

Copyright Node.js contributors. All rights reserved.

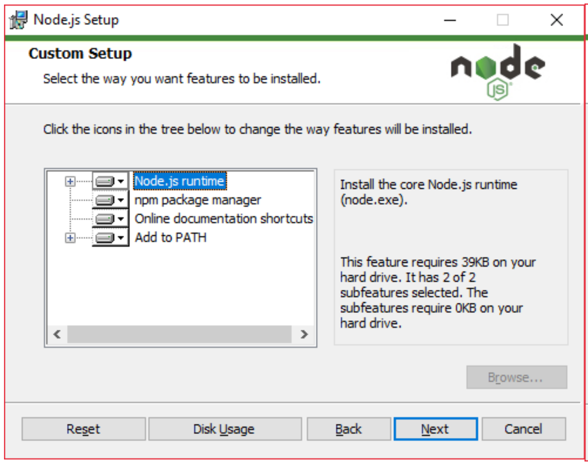
Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject



Install Node.js to:

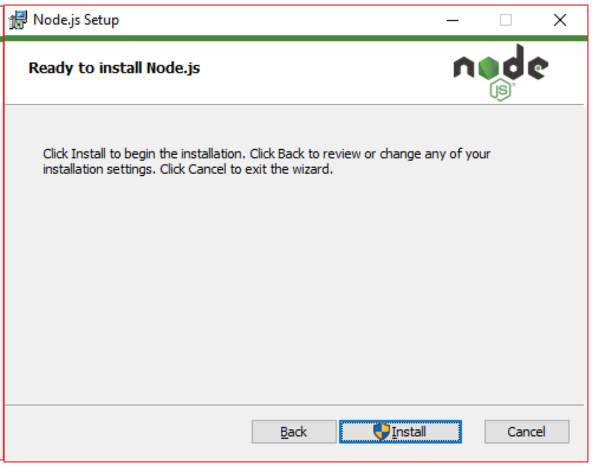
C:\Program Files\nodejs\

Select the Destination path, keep the default program files location.


Node.js Setup
Custom Setup
Select the way you want features to be installed.
Click the icons in the tree below to change the way features will be installed.
Node.js runtime
npm package manager
Online documentation shortcuts
Add to PATH
Browse...
Reset
Disk Usage
Back
Next
Cancel

Install the core Node.js runtime (node.exe).

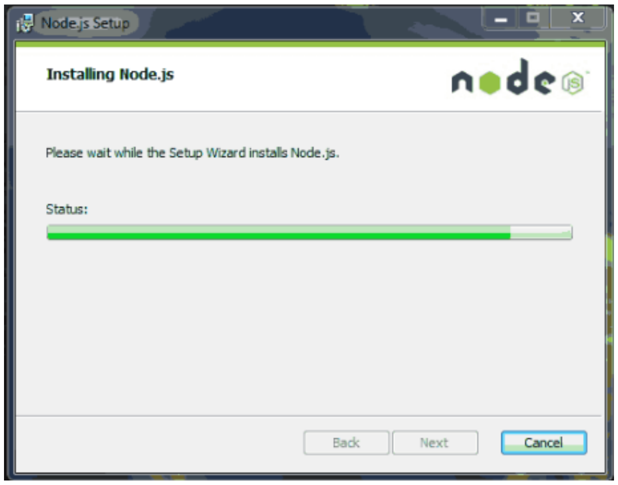
This feature requires 39KB on your hard drive. It has 2 of 2 subfeatures selected. The subfeatures require 0KB on your hard drive.



Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.

No changes in the custom settings , just click on next

Click on Install button, it installs the node js in your machine.

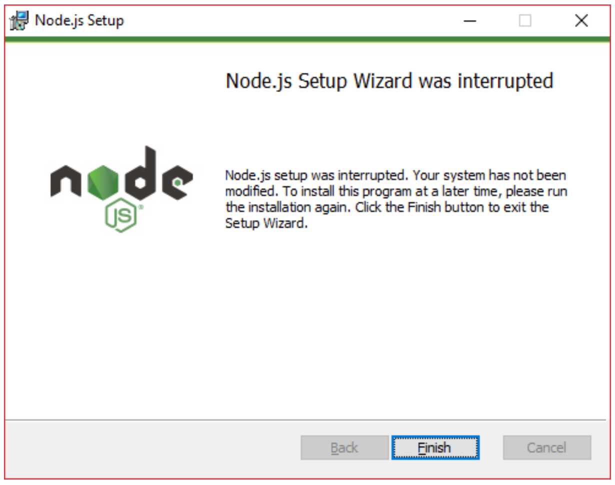


Node.js Setup

Installing Node.js

Please wait while the Setup Wizard installs Node.js.

Status:



Node.js Setup

Node.js Setup Wizard was interrupted

Node.js setup was interrupted. Your system has not been modified. To install this program at a later time, please run the installation again. Click the Finish button to exit the Setup Wizard.

Now we can verify the node JS version in the command prompt.

```
C:\Users\NAVEEN SAGGAM>node -v
v10.14.2

C:\Users\NAVEEN SAGGAM>
```

Node.js Console - REPL

Node.js comes with virtual environment called REPL (aka Node shell). REPL stands for Read-Eval-Print-Loop. It is a quick and easy way to test simple Node.js/JavaScript code.

REPL stands for Read Eval Print Loop and it represents a computer environment like a Windows console or Unix/Linux shell where a command is entered and the system responds with an output in an interactive mode. Node.js or Node comes bundled with a REPL environment. It performs the following tasks:

● Read - Reads user's input, parses the input into JavaScript data-structure, and stores in memory.

● Eval - Takes and evaluates the data structure.

● Print - Prints the result.

● Loop - Loops the above command until the user presses ctrl-c twice. The REPL feature of Node is very useful in experimenting with Node.js codes and to debug JavaScript codes

To launch the REPL (Node shell), open command prompt (in Windows) or terminal (in Mac or UNIX/Linux) and type node as shown below. It will change the prompt to > in Windows and MAC.

```
C:\Users\NAVEEN SAGGAM>node
>
```

Simple Expressions

```
C:\Users\NAVEEN SAGGAM>node
> 10 + 20
30
>
```

Use variables

You can make use variables to store values and print later like any conventional script. If var keyword is not used, then the value is stored in the variable and printed. Whereas if var keyword is used, then the value is stored but not printed. You can print variables using console.log().

```
C:\Users\NAVEEN SAGGAM>node
> let x = 10;
undefined
> let y = 20;
undefined
> console.log(`The Sum of x , y is : ${x + y}`);
The Sum of x , y is : 30
undefined
>
```

Multiline Expression

Node REPL supports multiline expression similar to JavaScript. Let's check the following dowhile loop in action:

```
C:\Users\NAVEEN SAGGAM>node
> let number = 10;
undefined
> let output = '';
undefined
> for(let i=0; i<=number; i++){
... output += `${i} `;
... }
'0 1 2 3 4 5 6 7 8 9 10 '
>
```

REPL Commands

- ctrl + c - terminate the current command.
- ctrl + c twice - terminate the Node REPL.
- ctrl + d - terminate the Node REPL.
- Up/Down Keys - see command history and modify previous commands.
- tab Keys - list of current commands.
- .help - list of all commands.
- .break - exit from multiline expression.
- .clear - exit from multiline expression.
- .save filename - save the current Node REPL session to a file.
- .load filename - load file content in current Node REPL session.

Node.js Basics

Node.js supports JavaScript. So, JavaScript syntax on Node.js is similar to the browser's JavaScript syntax.

Primitive Types

Node.js includes following primitive types:

1. String
2. Number
3. Boolean
4. Undefined
5. Null

Everything else is an object in Node.js.

Loose Typing

JavaScript in Node.js supports loose typing like the browser's JavaScript. Use let keyword to declare a variable of any type.

```
let currentCourse = 'nodejs';
let version = 10.5;
let usingEditor = true;
```

Object Literal

Object literal syntax is same as browser's JavaScript.

```
// Create JavaScript Object
let employee = {
    name : 'John',
    age : 40,
    designation : 'Manager',
    address : {
        city : 'Hyderabad',
        state : 'TS',
        country : 'India'
    }
};
console.log(employee);
```

Functions

Functions are first class citizens in Node's JavaScript, similar to the browser's JavaScript. A function can have attributes and properties also. It can be treated like a class in JavaScript.

```
// Normal Function
function printEmployee_1(employee) {
    console.log(employee);
}

// Function Expression
let printEmployee_2 = function(employee) {
    console.log(employee);
};

// Arrow Function from ES6
let printEmployee_3 = (employee) => {
    console.log(employee);
};
```

Buffer

Node.js includes an additional data type called Buffer (not available in browser's JavaScript). Buffer is mainly used to store binary data, while reading from a file or receiving packets over the network.

process object

Each Node.js script runs in a process. It includes process object to get all the information about the current process of Node.js application.

The following example shows how to get process information in REPL using process object.

Defaults to local

Node's JavaScript is different from browser's JavaScript when it comes to global scope. In the browser's JavaScript, variables declared without var keyword become global. In Node.js, everything becomes local by default.

Access Global Scope

In a browser, global scope is the window object. In Node.js, global object represents the global scope.

To add something in global scope, you need to export it using export or module.export. The same way, import modules/object using require() function to access it from the global scope.

For example, to export an object in Node.js, use exports.name = object.

Now, you can import log object using require() function and use it anywhere in your Node.js project.

Learn about modules in detail in the next section.

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.

Node.js Module Types

Node.js includes three types of modules:

1. Core Modules
2. Custom / Local Modules
3. 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.

Core Module	Description
http	http module includes classes, methods and events to create Node.js http server.
https	Same as http module , with secured SSL server creations
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.
os	Os module deals with accessing the underlying operating system properties.

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

```
const 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

```
const os = require('os'); // OS module

let totalMem = os.totalmem();
console.log(`Total Memory : ${totalMem}`);

let freeMem = os.freemem();
console.log(`Free Memory : ${freeMem}`);

let hostName = os.hostname();
console.log(`HostName : ${hostName}`);

let userInfo = os.userInfo().username;
console.log(`UserName : ${userInfo}`);
```

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.

Node.js custom / Local Modules

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.

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.

```
let 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 // to export the module outside
};
```

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.

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.

Now, let's see how to use the above logging module in our application.

```
const log = require('./log');

log.info('This is an info message');
log.warning('This is an info message');
log.error('This is an info message');
```

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.

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' or './log' 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 log.info() or log.warning() or log.error()

Export Module in Node.js

In the previous section, you learned how to write a local module using module.exports. In this section, you will learn how to expose different types as a module using module.exports.

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

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

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.

```
module.exports = 'Good Morning';
```

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

```
const msg = require('./messages');
console.log(msg);
```

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

Export Object

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.

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

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

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

The above module will expose an object

```
// app.js
const msg = require('./log.js');
msg.log('Hello World');
```

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

```
// data.js
module.exports = {
    firstName: 'John',
    lastName: 'Wilson'
}
```

```
// app.js
const person = require('./data.js');
console.log(person.firstName + ' ' + person.lastName);
```

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 below.

```
// app.js
const msg = require('./Log.js');
msg('Hello World');
```

The msg variable becomes 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.

Load Module from Separate Folder

Use the full path of a module file where you have exported it using module.exports. For example, if log module in the log.js is stored under "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 root folder and then specify exact path of your module file. Node.js also allows us to specify the path to the folder without specifying file name.

Node Package Manager

Node Package Manager (NPM) is a command line tool that installs, updates or uninstalls Node.js packages in your application. It is also an online repository for open-source Node.js packages. The node community around the world creates useful modules and publishes them as packages in this repository..

Official website: https://www.npmjs.com

NPM is included with Node.js installation. After you install Node.js, verify NPM installation by writing the following command in terminal or command prompt.

```
C:\Users>node -v
v10.15.1
```

If you have an older version of NPM then you can update it to the latest version using the following command.

```
C:\Users>npm i npm -g
```

To access NPM help, write npm help in the command prompt or terminal window.

```
C:\Users>npm help
```

NPM performs the operation in two modes: global and local. In the global mode, NPM performs operations which affect all the Node.js applications on the computer whereas in the local mode, NPM performs operations for the particular local directory which affects an application in that directory only.

Install Package Locally

Use the following command to install any third party module in your local Node.js project folder.

```
C:\Users>npm install <package_name>
```

For example, the following command will install ExpressJS into Project1 folder.

```
C:\project_1>npm install express
```

All the modules installed using NPM are installed under node_modules folder. The above command will create ExpressJS folder under node_modules folder in the root folder of your project and install Express.js there.

Add Dependency into package.json

Use --save at the end of the install command to add dependency entry into package.json of your application.

For example, the following command will install ExpressJS in your application and also adds dependency entry into the package.json.

```
C:\project_1>npm install express --save
```

The package.json of Node JS project will look something like below.

```
// package.json
{
  "name": "express_app",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.16.4"
  }
}
```

Install Package Globally

NPM can also install packages globally so that all the node.js application on that computer can import and use the installed packages. NPM installs global packages into /<User>/local/lib/node_modules folder.

Apply -g in the install command to install package globally. For example, the following command will install ExpressJS globally.

```
C:\project_1>npm install -g express
```

Update Package

To update the package installed locally in your Node.js project, navigate the command prompt or terminal window path to the project folder and write the following update command.

```
C:\project_1>npm update <package_name>
```

The following command will update the existing ExpressJS module to the latest version.

```
C:\project_1>npm update express
```

Uninstall Packages

Use the following command to remove a local package from your project.

```
C:\project_1>npm uninstall <package_name>
```

The following command will uninstall Express JS from the application.

```
C:\project_1>npm uninstall express
```

Node JS Webserver

In this section, we will learn how to create a simple Node.js web server and handle HTTP requests.

In this section, we will learn how to create a simple Node.js web server and handle HTTP requests.

To access web pages of any web application, you need a 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

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.

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 **app.js** file.

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

const hostname = '127.0.0.1';
const port = 3000;

// 2 - creating server
const server = http.createServer((request, response) => {
    //handle incomming requests here..
});
```

```
//3 - listen for any incoming requests
server.listen(port,hostname, () => {
    console.log(`Server is started at http://${hostname}:${port}`);
});
```

In the above example, we import the http module using require() function.

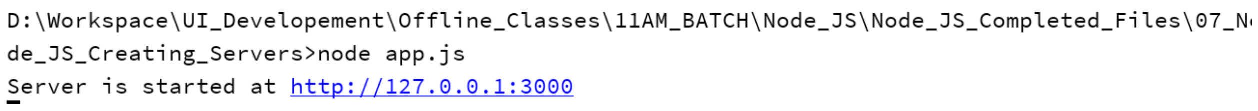
The http module is a core module of Node.js, so no need to install it using NPM.

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 3000.

Note: You can specify any unused port here.

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



```
D:\Workspace\UI_Developement\Offline_Classes\11AM_BATCH\Node_JS\Node_JS_Completed_Files\07_N
de_JS_Creating_Servers>node app.js
Server is started at http://127.0.0.1:3000
```

Handle HTTP Request

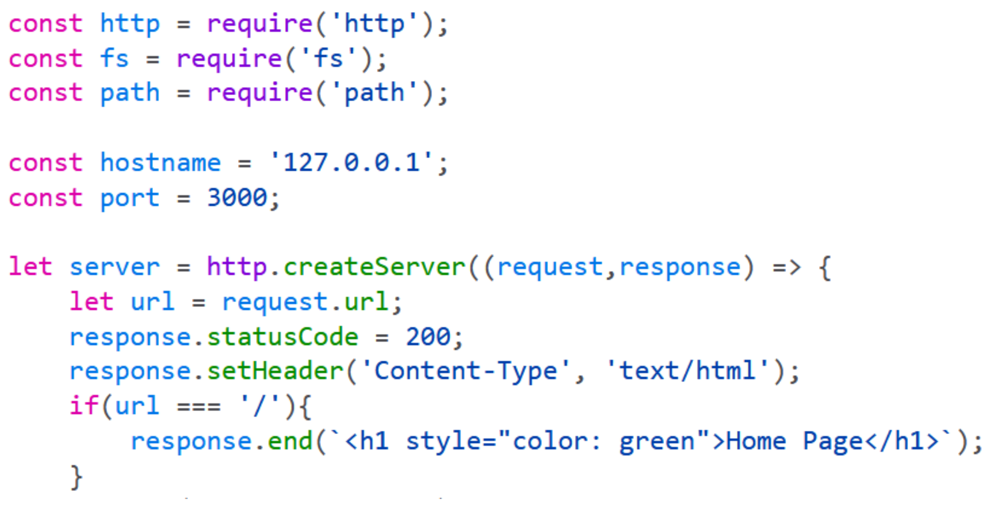
The http.createServer() method includes request and response parameters which is supplied by Node.js.

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



```
const http = require('http');
const fs = require('fs');
const path = require('path');

const hostname = '127.0.0.1';
const port = 3000;

let server = http.createServer((request,response) => {
    let url = request.url;
    response.statusCode = 200;
    response.setHeader('Content-Type', 'text/html');
    if(url === '/'){
        response.end(`<h1 style="color: green">Home Page</h1>`);
    }
    else if(url === '/about'){
        response.end(`<h1 style="color: green">About Page</h1>`);
    }
    else if(url === '/services'){
        response.end(`<h1 style="color: green">Services Page</h1>`);
    }
    else if(url === '/contact'){
        response.end(`<h1 style="color: green">Contact Page</h1>`);
    }
    else{
        response.end(`<h1 style="color: red">Page Not Found</h1>`);
    }
});

server.listen(port,hostname, () => {
    console.log(`Server is started at http://${hostname}:${port}`);
});
```

In the above example, request.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 setHeader() method. Finally, Node.js web server sends the response using end() method.

Now, run the above web server as shown below.

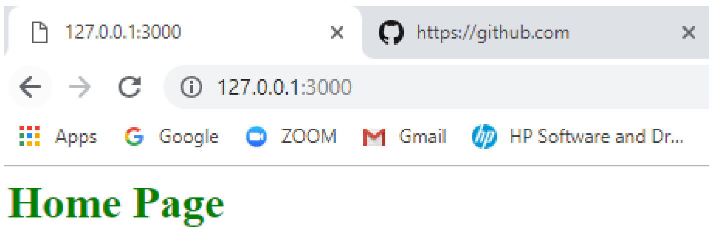
```
D:\Workspace\UI_Developement\Offline_Classes\11AM_BATCH\Node_JS\Node_JS_Completed_Files\07_No
de_JS_Creating_Servers>node app.js
Server is started at http://127.0.0.1:3000
```

You can test the connection in the new command prompt

```
curl -i http://127.0.0.1:3000
```

```
c:\> curl -i http://127.0.0.1:3000
HTTP/1.1 200 OK
Content-Type: text/html
Date: Tue, 26 Feb 2019 11:13:22 GMT
Connection: keep-alive
Content-Length: 39
<h1 style="color: green">Home Page</h1>
```

For Windows users, point your browser to http://localhost:3000 and see the following result.

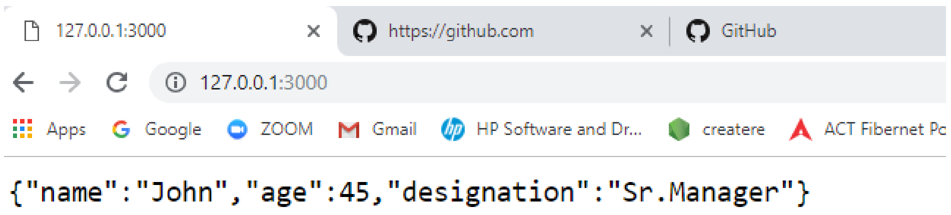


Home Page

Sending JSON Response

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

```
let server = http.createServer((request,response) => {
    let url = request.url;
    response.statusCode = 200;
    response.setHeader('Content-Type', 'application/json');
    let employee = {
        name : 'John',
        age : 45,
        designation : 'Sr.Manager'
    };
    if(url === '/'){
        response.write(JSON.stringify(employee));
        response.end();
    }
});
```


127.0.0.1:3000
https://github.com
GitHub
127.0.0.1:3000
Apps Google ZOOM Gmail HP Software and Dr... createre ACT Fibernet Pc

```
{"name":"John","age":45,"designation":"Sr.Manager"}
```

Node.js File System

Node.js includes fs module to access physical file system. The fs module is responsible for all the asynchronous or synchronous file I/O operations.

Let's see some of the common I/O operation examples using fs module.

Reading File

Use fs.readFile() method to read the physical file asynchronously.

```
fs.readFile(path[, options], callback)
```

Parameter Description:

filename: Full path and name of the file as a string.

options: The options parameter can be an object or string which can include encoding and flag. The default encoding is utf8 and default flag is "r".

callback: A function with two parameters err and fd. This will get called when readFile operation completes.

The following example demonstrates reading existing data.txt asynchronously.

```
fs.readFile('data.txt','utf8',(err,data) => {
    if(err) throw err;
    console.log(data);
});
```

The above example reads data.txt (on Windows) asynchronously and executes callback function when read operation completes.

This read operation either throws an error or completes successfully. The err parameter contains error information if any. The data parameter contains the content of the specified file.

Writing File

Use fs.writeFile() method to write data to a file. If file already exists then it overwrites the existing content otherwise it creates a new file and writes data into it.

```
fs.writeFile(filename, data[, options], callback);
```

Parameter Description:

filename: Full path and name of the file as a string.

Data: The content to be written in a file.

options: The options parameter can be an object or string which can include encoding, mode and flag. The default encoding is utf8 and default flag is "r".

callback: A function with two parameters err and fd. This will get called when write operation completes.

```
fs.writeFile('data.txt','Good Morning','utf8',(err) => {
    if(err) throw err;
    console.log('Data added to a file');
});
```

In the same way, use fs.appendFile() method to append the content to an existing file.

```
const fs = require('fs');
fs.appendFile('data.txt', 'Hello World!', function (err) {
    if(err) throw err;
    console.log('Append operation complete.');
});
```

Delete File

Use fs.unlink() method to delete an existing file.

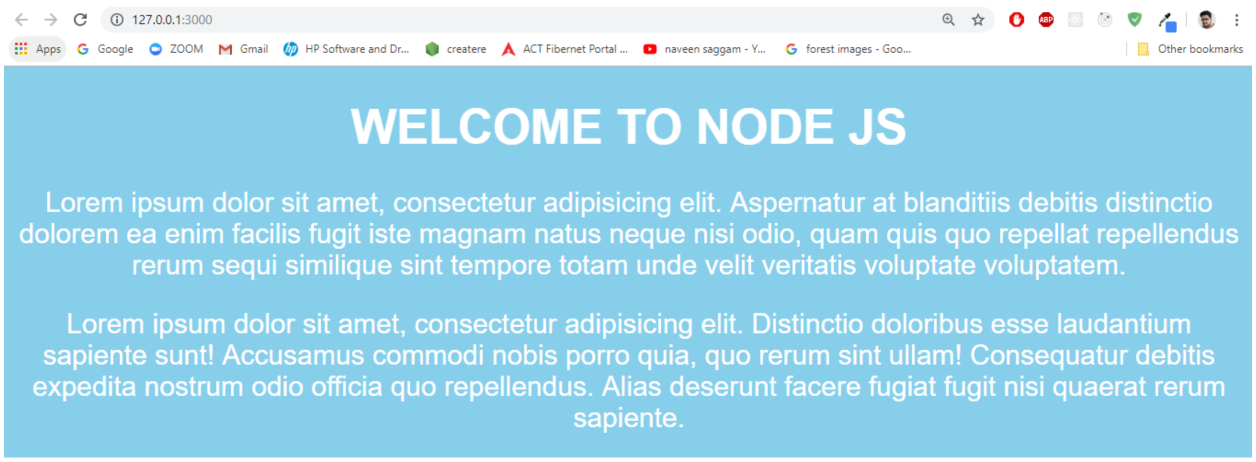
```
fs.unlink(path, callback);
```

```
const fs = require('fs');
fs.unlink('data.txt', function () {
     console.log('Deleted the file.');
});
```

Display HTML Content

We can display the complete HTML Content on the browser using the file system module is as follows,

```javascript
let server = http.createServer((request,response) => {
    let url = request.url;
    console.log(url);
    response.statusCode = 200;
    response.setHeader('Content-Type', 'text/html');
    if(url === '/'){
        fs.readFile(path.join(__dirname,'home.html'),'utf8',(err,data) => {
            response.end(data);
        });
    }
    else{
        response.end(`<h1 style="color: red">Page Not Found</h1>`);
    }
});
```



WELCOME TO NODE JS

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aspernatur at blanditiis debitis distinctio dolorem ea enim facilis fugit iste magnam natus neque nisi odio, quam quis quo repellat repellendus rerum sequi similique sint tempore totam unde velit veritatis voluptate voluptatem.

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Distinctio doloribus esse laudantium sapiente sunt! Accusamus commodi nobis porro quia, quo rerum sint ullam! Consequatur debitis expedita nostrum odio officia quo repellendus. Alias deserunt facere fugiat fugit nisi quaerat rerum sapiente.

Node JS Routing

We can construct the routing system using Node JS. In this routing we can display various html pages based on the user provided url.

We can even generate the response based on the clients get requests.

Let’s understand the routing process in code Node JS.

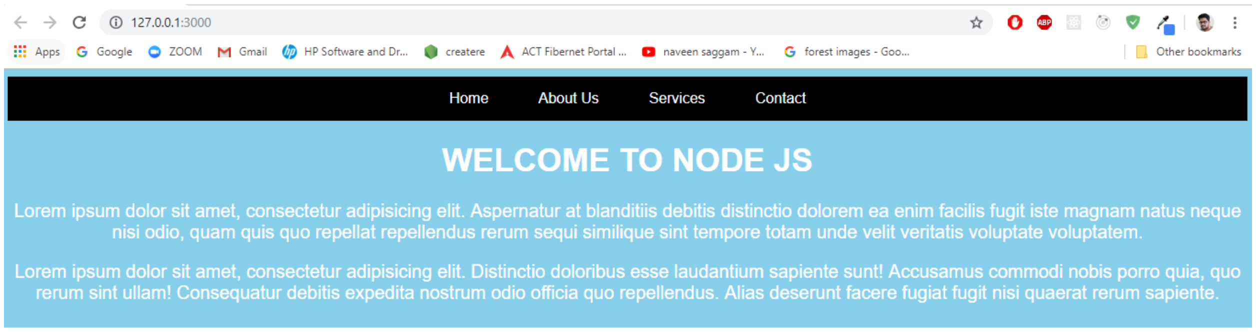
Create any 4 html files with some basic data in it.

And refer them using node js code as follows,

```javascript
let server = http.createServer((request,response) => {
    let url = request.url;
    console.log(url); // get Request URL
    response.statusCode = 200;
    response.setHeader('Content-Type', 'text/html');
    if(url === '/'){
        fs.readFile(path.join(__dirname,'home.html'),'utf8',(err,data) => {
            response.end(data);
        });
    }
    else if(url === '/about'){
        fs.readFile(path.join(__dirname,'about.ejs'),'utf8',(err,data) => {
            response.end(data);
        });
    }
    else if(url === '/services'){
        fs.readFile(path.join(__dirname,'services.ejs'),'utf8',(err,data) => {
            response.end(data);
        });
    }
    else if(url === '/contact'){
        fs.readFile(path.join(__dirname,'contact.ejs'),'utf8',(err,data) => {
            response.end(data);
        });
    }
    else{
        response.end(`<h1 style="color: red">Page Not Found</h1>`);
    }
});

server.listen(port,hostname,() => {
    console.log(`Server running at http://${hostname}:${port}`);
});
```

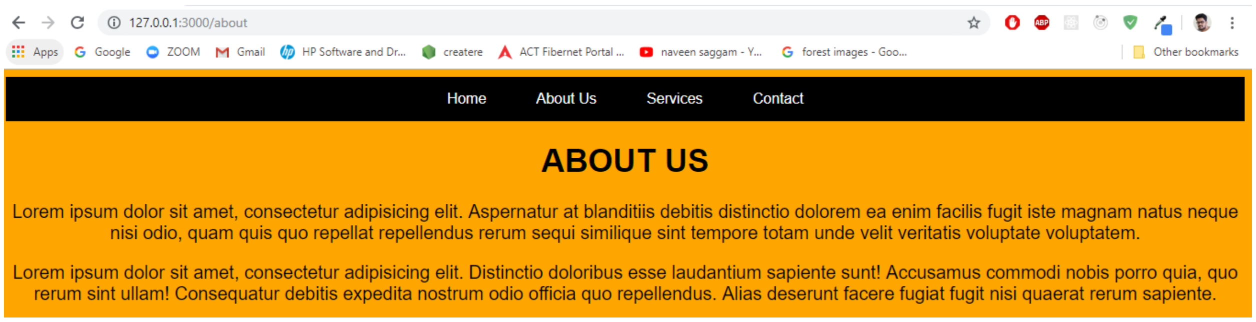
The Output of this node js routing will be like as follows,



WELCOME TO NODE JS

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aspernatur at blanditiis debitis distinctio dolorem ea enim facilis fugit iste magnam natus neque nisi odio, quam quis quo repellat repellendus rerum sequi similique sint tempore totam unde velit veritatis voluptate voluptatem.

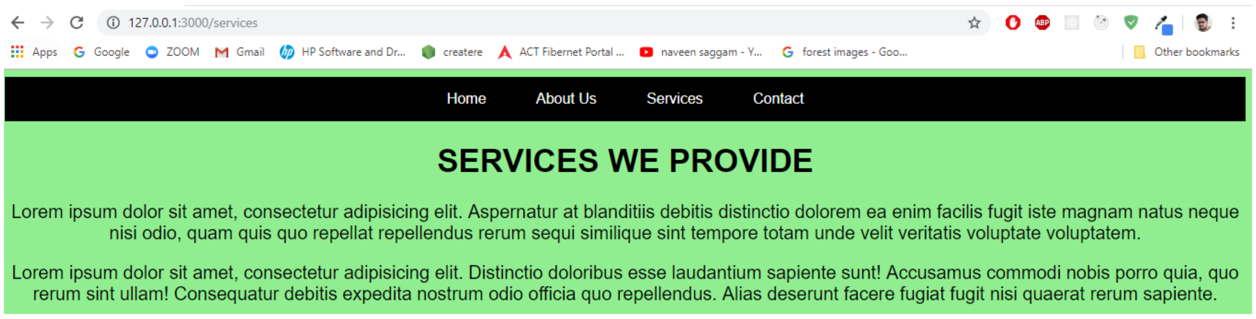
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Distinctio doloribus esse laudantium sapiente sunt! Accusamus commodi nobis porro quia, quo rerum sint ullam! Consequatur debitis expedita nostrum odio officia quo repellendus. Alias deserunt facere fugiat fugit nisi quaerat rerum sapiente.



ABOUT US

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aspernatur at blanditiis debitis distinctio dolorem ea enim facilis fugit iste magnam natus neque nisi odio, quam quis quo repellat repellendus rerum sequi similique sint tempore totam unde velit veritatis voluptate voluptatem.

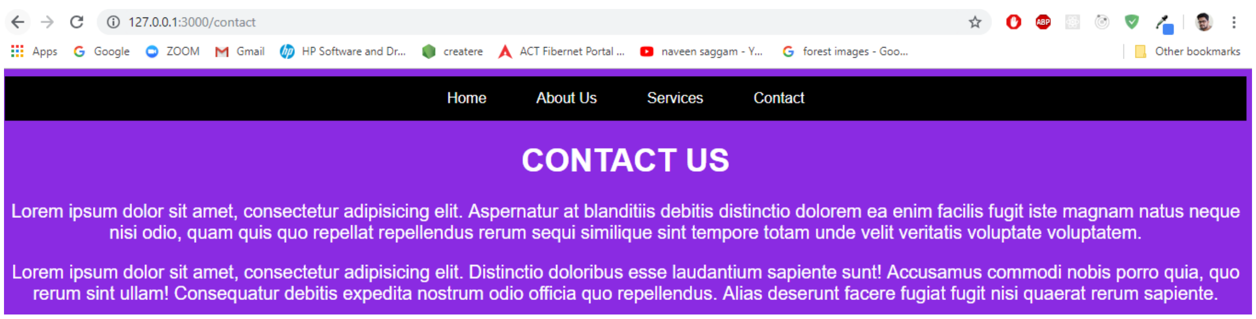
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Distinctio doloribus esse laudantium sapiente sunt! Accusamus commodi nobis porro quia, quo rerum sint ullam! Consequatur debitis expedita nostrum odio officia quo repellendus. Alias deserunt facere fugiat fugit nisi quaerat rerum sapiente.



SERVICES WE PROVIDE

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aspernatur at blanditiis debitis distinctio dolorem ea enim facilis fugit iste magnam natus neque nisi odio, quam quis quo repellat repellendus rerum sequi similique sint tempore totam unde velit veritatis voluptate voluptatem.

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Distinctio doloribus esse laudantium sapiente sunt! Accusamus commodi nobis porro quia, quo rerum sint ullam! Consequatur debitis expedita nostrum odio officia quo repellendus. Alias deserunt facere fugiat fugit nisi quaerat rerum sapiente.



CONTACT US

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aspernatur at blanditiis debitis distinctio dolorem ea enim facilis fugit iste magnam natus neque nisi odio, quam quis quo repellat repellendus rerum sequi similique sint tempore totam unde velit veritatis voluptate voluptatem.

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Distinctio doloribus esse laudantium sapiente sunt! Accusamus commodi nobis porro quia, quo rerum sint ullam! Consequatur debitis expedita nostrum odio officia quo repellendus. Alias deserunt facere fugiat fugit nisi quaerat rerum sapiente.

Frameworks for Node.js

You learned that we need to write lots of low level code ourselves to create a web application using Node.js in Node.js web server section.

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.

Framework	Website	Description
Express JS	http://expressjs.com/	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	http://geddyjs.org/	Geddy is a simple, structured web application framework for Node.js based on MVC architecture.
Locomotive	http://www.locomotivejs.org/	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.

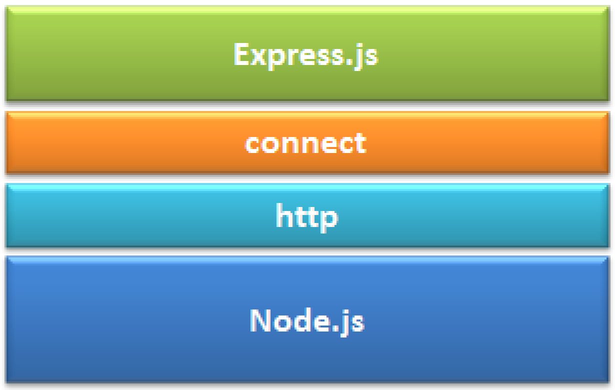
Let's learn about Express JS Now.

Express.js

"Express is a fast, un-opinionated 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.



Express.js

Advantages of Express.js

Makes Node.js web application development fast and easy.

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

Install Express.js

You can install express.js using npm. The following command will install latest version of express.js globally on your machine so that every Node.js application on your machine can use it.

```
npm install -g express
```

The following command will install latest version of express.js local to your project folder.

```
npm install express --save
```

As you know, --save will update the package.json file by specifying express.js dependency.

Express.js Web Application

In this section, you will learn how to create a web application using Express.js.

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.

```
const express = require('express');
const app = express();
const hostname = '127.0.0.1';
const port = 3000;

app.get('/', (req, res) => res.send('Hello World!'));
app.listen(port,hostname,() => {
    console.log(`Server running at http://${hostname}:${port}`);
});
```

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:3000. It will display Cannot GET / because we have not configured any routes yet.

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.

```
app.get('/', function (req, res) {
    res.send('<html><body><h1>Hello World</h1></body></html>');
});

// POST Request
app.post('/submit-data', function (req, res) {
    res.send('POST Request');
});

// PUT Request
app.put('/update-data', function (req, res) {
    res.send('PUT Request');
});

// DELETE Request
app.delete('/delete-data', function (req, res) {
    res.send('DELETE Request');
});
```

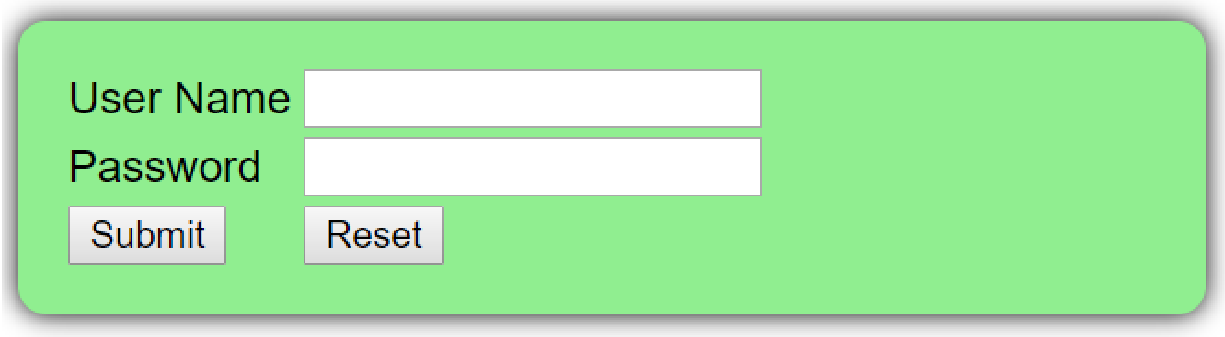
In the above example, app.get(), app.post(), app.put() and app.delete() methods define routes for HTTP GET, POST, PUT, DELETE respectively. The first parameter is a path of a route which will start after base URL. The callback function includes request and response object which will be executed on each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

Run the above example using node server.js command, and point your browser to http://127.0.0.1:3000 and you will see the following result.

Handle POST Request

Here, you will learn how to handle HTTP POST request and get data from the submitted form.

First, create index.html file in the root folder of your application and write the following HTML code in it.



```
<form>
    <label>UserName</label>
    <input type="text">
    <br>
    <label>Password</label>
    <input type="password">
    <br>
    <input type="submit" value="Submit">
    <input type="reset" value="Reset">
</form>
```

To handle HTTP POST request in Express.js version 4 and above, you need to install middleware module called **body-parser**. The middleware was a part of Express.js earlier but now you have to install it separately.

This body-parser module parses the JSON, buffer, string and url encoded data submitted using HTTP POST request. Install body-parser using NPM as shown below.

```
npm install body-parser --save
```

Now, import body-parser and get the POST request data as shown below.

```
const bodyParser = require("body-parser");
app.use(bodyParser.urlencoded({ extended: false }));

app.post('/submit-student-data', function (req, res) {
    let username = req.body.username;
    let password = req.body.password
    res.send(' Submitted Successfully!');
});
```

In the above example, POST data can be accessed using req.body. The req.body is an object that includes properties for each submitted form. Index.html contains username and password input types, so you can access it using req.body.username and req.body.password.

Now, run the above example using node server.js command, point your browser to http://127.0.0.1:3000 and see the following result.

Serving Static Resources in Node.js

In this section, you will learn how to serve static resources like images, css, JavaScript or other static files using Express.js and node-static module.

Serve Static Resources using Express.js

It is easy to serve static files using built-in middleware in Express.js called express.static. Using express.static() method, you can server static resources directly by specifying the folder name where you have stored your static resources.

The following example serves static resources from the public folder under the root folder of your application.

```
//setting middleware
app.use('/public', express.static(__dirname + 'public'));
```

In the above example, app.use() method mounts the middleware express.static for every request. The express.static middleware is responsible for serving the static assets of an Express.js application. The express.static() method specifies the folder from which to serve all static resources.

Now, run the above code using node server.js command and point your browser to http://127.0.0.1:3000/myImage.jpg and it will display myImage.jpg from the public folder (public folder should have myImage.jpg).

If you have different folders for different types of resources then you can set express.static middleware as shown below.

```
//Serves all the request which includes /images in the url from Images folder
app.use('/images', express.static(__dirname + '/Images'));
```

In the above example, app.use() method mounts the express.static middleware for every request that starts with "/images". It will serve images from images folder for every HTTP requests that starts with "/images". For example, HTTP request http://127.0.0.1:3000/images/myImage.png will get myImage.png as a response. All other resources will be served from public folder.

Now, run the above code using node server.js and point your browser to http://127.0.0.1:3000/images/myImage.jpg and it will display myImage.jpg from the images folder, whereas http://127.0.0.1:3000/myJSFile.js request will be served from public folder. (images folder must include myImage.png and public folder must include myJSFile.js)

You can also create a virtual path in case you don't want to show actual folder name in the url.

```
// Setting Virtual Path
app.use('/resources',express.static(__dirname + '/images'));
```

So now, you can use http://127.0.0.1:3000/resources/myImage.jpg to serve all the images instead of http://127.0.0.1:3000/images/myImage.jpg.

In this way, you can use Express.js to server static resources such as images, CSS, JavaScript or other files.

Data Access in Node.js

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.

Relational Database	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

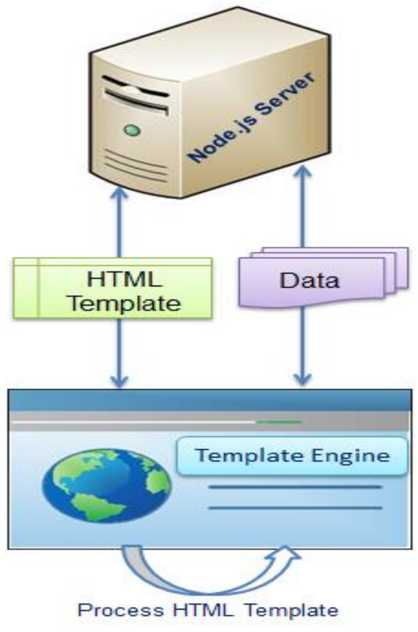
The following table lists important NoSQL databases and respective drives .

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

Template Engines for Node.js

Template engine helps us to create an HTML template with minimal code. Also, it can inject data into HTML template at client side and produce the final HTML.

The following figure illustrates how template engine works in Node.js.



Template Engine

As per the above figure, client-side browser loads HTML template, JSON/XML data and template engine library from the server. Template engine produces the final HTML using template and data in client's browser. However, some HTML templates process data and generate final HTML page at server side also.

There are many template engines available for Node.js. Each template engine uses a different language to define HTML template and inject data into it.

The following is a list of important (but not limited) template engines for Node.js

Template Engine	Official Website
EJS	https://ejs.co/
PUG / JADE	https://pugjs.org/api/getting-started.html
Handlebars / Mustache	https://handlebarsjs.com/
Vash Template	https://github.com/kirbysayshi/vash
Dust JS	http://www.dustjs.com/
NunJucks	https://mozilla.github.io/nunjucks/
Haml Template Engine	http://haml.info/

Advantages of Template Engine

1) Improves developer's productivity.
2) Improves readability and maintainability.
3) Faster performance.
4) Maximizes client side processing.
5) Single template for multiple pages.
6) Templates can be accessed from CDN (Content Delivery Network).

EJS (Embedded JS) Template Engine

This Template Engine is used to display any dynamic data from the server to the browser.

In this, we won't use DOM manipulation because the data has to serve from the server.

In this template Engine, we mix the JavaScript code with the HTML Tags. with EJS Syntax.

```
<% if (user) { %>
  <h2><%= user.name %></h2>
<% } %>
```

Each file which is serving dynamic data from the server using EJS should be have an extension '.ejs'

Ex: index.ejs , about.ejs , services.ejs etc

To use any Template engine in Express js application, we have to add a middle ware to express js as follows,

```
// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');
```

Mostly all the Template Engine files are to be kept in the 'views' folder. This is the default folder location the express js looks for ejs files / template engine specific files. Other than this folder we have to mention in the middleware.

To send any response back to browser in the form of ejs file, we use the 'render()' function to send the ejs / template engine data.

```
// for Home Page
router.get('/', function(req, res, next) {
  res.render('index', { pageTitle: 'Express EJS' , page : 'home'});
});

// for About Us Page
router.get('/about', function(req, res, next) {
    res.render('about', { pageTitle: 'About Us' ,page : 'about' });
});
```

Any data/objects to be send to the front end, we have to send them using the above syntax like pageTitle and page parameters.

We can access this data / parameters / Objects on the frontend ejs file using the ejs file syntax as follows,

```
// Index.ejs
<title><%= pageTitle %></title>

<h1 class="text-primary"><%= pageTitle %></h1>
```

```
<li class="nav-item <%= (page === 'home')? 'active' : '' %>">
    <a class="nav-link" href="/">Home</a>
</li>
```

Loading partial files in EJS

We can make a separate files for any specific logic / common logic / template and we can import / include it in the existing template to reuse the existing ejs file in any other ejs file as follows.

We mostly use this approach of loading partials for main Navigation bar , Footer files or any special components to reuse.

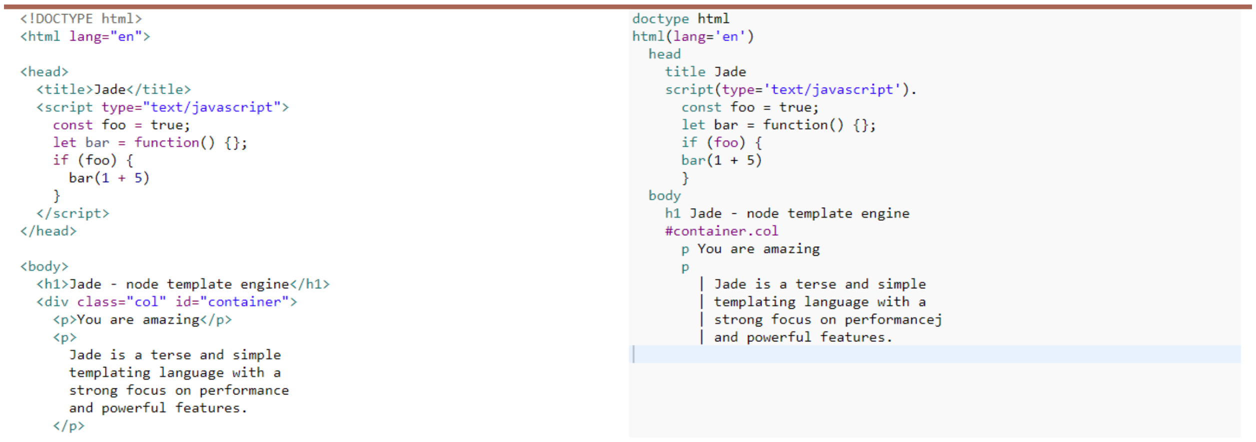
```
<!-- Main Navigation -->
<% include layout/navbar.ejs %>
```

PUG / JADE Template Engine

Pug Template Engine is also an option for your Node JS Application. This is a quite a different way of making even HTML Template.

This is having the file Extension of '.pug. like index.pug , about.pug etc.

Let's see the different between HTML page and Pug template as follows,



```
<!DOCTYPE html>
<html lang="en">

<head>
  <title>Jade</title>
  <script type="text/javascript">
    const foo = true;
    let bar = function() {};
    if (foo) {
      bar(1 + 5)
    }
  </script>
</head>

<body>
  <h1>Jade - node template engine</h1>
  <div class="col" id="container">
    <p>You are amazing</p>
    <p>
      Jade is a terse and simple
      templating language with a
      strong focus on performance
      and powerful features.
    </p>
```

```
doctype html
html(lang='en')
  head
    title Jade
    script(type='text/javascript').
      const foo = true;
      let bar = function() {};
      if (foo) {
      bar(1 + 5)
      }
  body
    h1 Jade - node template engine
    #container.col
      p You are amazing
      p
        | Jade is a terse and simple
        | templating language with a
        | strong focus on performancej
        | and powerful features.
```

This is the new template language where no close tags available.

If you are familiar with HTML language, there you can easily learn PUG template.

There are many online tools available for this PUG template Conversion such as,

https://html2jade.org/ OR https://html-to-pug.com/

In order to display any dynamic data with the template we use the following Syntax,

```
// for dynamic data / variable
#{pageTitle}

// for conditionals
if (page === 'home')
  li.nav-item.active
    a.nav-link(href='/') Home
else
  li.nav-item
    a.nav-link(href='/') Home
```

To Use this template Engine, we can define a middleware inside Express JS as follows,

```
// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'pug');
```

Mostly all the Template Engine files are to be kept in the 'views' folder. This is the default folder location the express js looks for pug files / template engine specific files. Other than this folder we have to mention in the middleware.

To send any response back to browser in the form of pug file, we use the 'render()' function to send the pug / template engine data.

```
/* GET About page. */
router.get('/about', function(req, res, next) {
    res.render('about', { pageTitle: 'About Us' , page : 'about' });
});

/* GET Contact Us page. */
router.get('/contact', function(req, res, next) {
    res.render('contact', { pageTitle: 'Contact Us' , page : 'contact'});
});
```

Loading partial files in PUG

We can make a separate files for any specific logic / common logic / template and we can import / include it in the existing template to reuse the existing pug file in any other pug file as follows.

We mostly use this approach of loading partials for main Navigation bar , Footer files or any special components to reuse.

```
// Main Navigation
include layout/navbar.pug
```

For More Information on this PUG Template Engine, you can refer the website,

```
https://pugjs.org/api/getting-started.html
```

Express Generator Tool

To Develop any node JS / Express JS Web application, Creating the Skelton of the application is the actual time consuming process.

In the manual way first we have to create the structure of the application means, the following structure.

```
Node_Application
   -> Views (for template engine files)
   -> routes (for routing purpose)
   -> app.js (main application file)
   -> database (for database specific files)
   -> util (for any utility files / supported files)
```

Instead of doing all these files creation we can use the express generator for creating the basic Project structure of us.

This is also a npm module, we can install this globally using the npm command as follows,

```
npm install express-generator -g
```

To create a Node JS / Express JS application using this Express JS Generator is as follows,

```
// syntax
express --view=<template_engine> <project_name>

// example
express --view=pug my_app
```

This command creates the following files and folders,

```
myapp
   | myapp/package.json
   | myapp/app.js
   | myapp/public
   | myapp/public/javascripts
   | myapp/public/images
   | myapp/routes
   | myapp/routes/index.js
   | myapp/routes/users.js
   | myapp/public/stylesheets
   | myapp/public/stylesheets/style.css
   | myapp/views
   | myapp/views/index.pug
   | myapp/views/layout.pug
   | myapp/views/error.pug
   | myapp/bin
   | myapp/bin/www
```

Modifications for Generated Files

There are some minor modification required for the files which are generated using Express Generator.

1) We need to modify the package.json file to use ‘nodemon’ module.

This is a module to automatically restart the server for every file change on the project.

This can be installed using npm as follows,

```
npm install -g nodemon
```

Change the package.json file to use nodemon as follows,

```
"scripts": {
  "start": "nodemon ./bin/www"
},
```

2) Replace all the ‘var’ keywords with ‘let’ or ‘const’ keywords.

This is because, this tool generates, ES5 Syntax keywords, we have to change this to ES6 keywords.

3) Change the static files middleware entry In the app.js file

```
app.use('/public', express.static('public'));
```

MongoDB is an open-source document database and leading NoSQL database. MongoDB is written in C++. This is a highly scalable and performance-oriented database.

MongoDB is a cross-platform, document oriented database that provides, high performance, high availability, and easy scalability. MongoDB works on concept of collection and document.

Database

Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

Collection

Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.

Document

A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

The following table shows the relationship of RDBMS terminology with MongoDB.

Relational Database	Mongo DB / No SQL Database
Database	Database
Table	Collection
Rows	Documents

Column	Field
Table Join	Embedded Document

Sample Document

Following example shows the document structure of a blog site, which is simply a comma separated key value pair.

```
{
   _id: ObjectId(7df78ad8902c)
   title: 'MongoDB Overview',
   description: 'MongoDB is no sql database',
   by: 'tutorials point',
   url: 'http://www.tutorialspoint.com',
   tags: ['mongodb', 'database', 'NoSQL'],
   likes: 100,
   comments: [
      {
         user:'user1',
         message: 'My first comment',
         dateCreated: new Date(2011,1,20,2,15),
         like: 0
      },
      {
         user:'user2',
         message: 'My second comments',
         dateCreated: new Date(2011,1,25,7,45),
         like: 5
      }
   ]
}
```

_id is a 12 bytes hexadecimal number which assures the uniqueness of every document. You can provide _id while inserting the document. If you don't provide then MongoDB provides a unique id for every document. These 12 bytes first 4 bytes for the current timestamp, next 3 bytes for machine id, next 2 bytes for process id of MongoDB server and remaining 3 bytes are simple incremental VALUE.

Any relational database has a typical schema design that shows number of tables and the relationship between these tables. While in MongoDB, there is no concept of relationship.

Advantages of MongoDB over RDBMS

1) Schema less − MongoDB is a document database in which one collection holds different documents. Number of fields, content and size of the document can differ from one document to another.
2) Structure of a single object is clear.
3) No complex joins.
4) Deep query-ability. MongoDB supports dynamic queries on documents using a document-based query language that's nearly as powerful as SQL.
5) Ease of scale-out − MongoDB is easy to scale.
6) Conversion/mapping of application objects to database objects not needed.

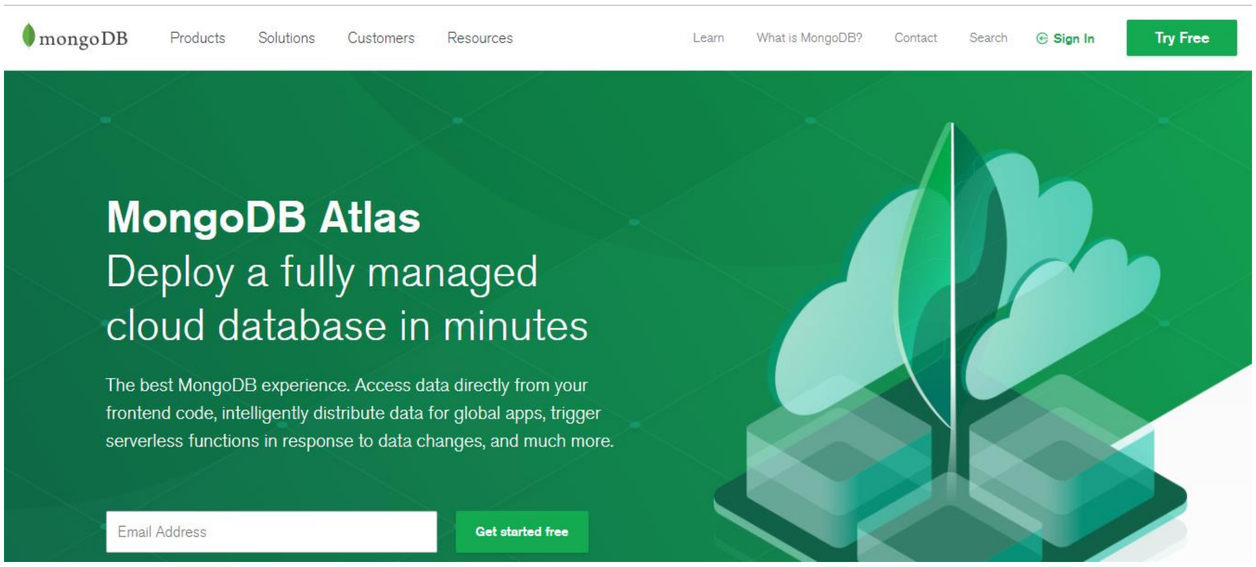
7) Uses internal memory for storing the (windowed) working set, enabling faster access of data.

Install MongoDB On Windows

To install MongoDB on Windows, first download the latest release of MongoDB from https://www.mongodb.org/downloads. Make sure you get correct version of MongoDB depending upon your Windows version

https://www.mongodb.com/

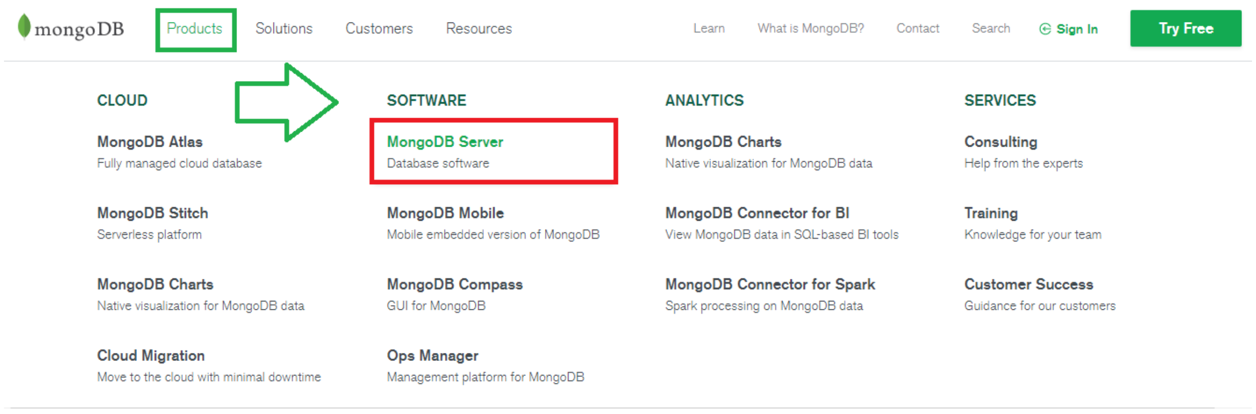
Go to the official website of mongo as mentioned above.



MongoDB Atlas
Deploy a fully managed cloud database in minutes

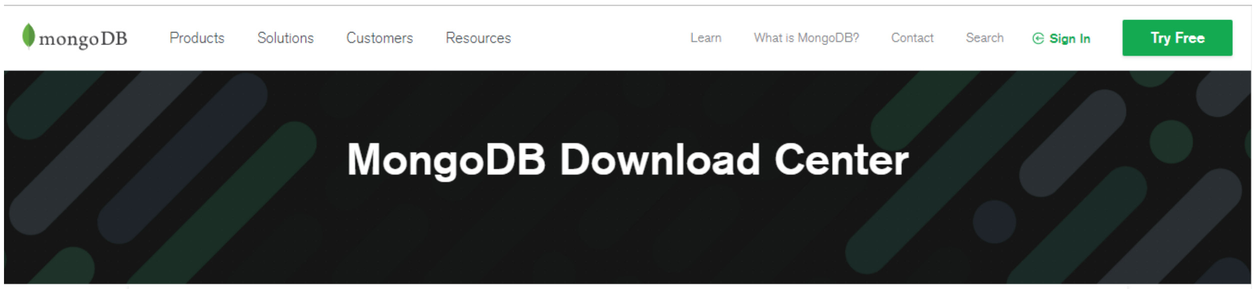
The best MongoDB experience. Access data directly from your frontend code, intelligently distribute data for global apps, trigger serverless functions in response to data changes, and much more.

Click on Products , and click on MongoDB Server.

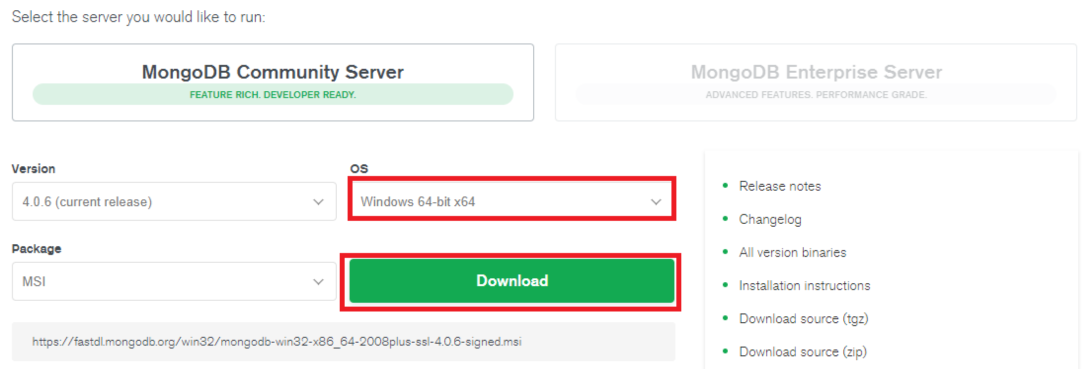


CLOUD	SOFTWARE	ANALYTICS	SERVICES
MongoDB Atlas Fully managed cloud database	MongoDB Server Database software	MongoDB Charts Native visualization for MongoDB data	Consulting Help from the experts
MongoDB Stitch Serverless platform	MongoDB Mobile Mobile embedded version of MongoDB	MongoDB Connector for BI View MongoDB data in SQL-based BI tools	Training Knowledge for your team
MongoDB Charts Native visualization for MongoDB data	MongoDB Compass GUI for MongoDB	MongoDB Connector for Spark Spark processing on MongoDB data	Customer Success Guidance for our customers
Cloud Migration Move to the cloud with minimal downtime	Ops Manager Management platform for MongoDB		

Once you click on this, this leads to Downloads Center.



In This Select the Operating system options and click on download.


Select the server you would like to run:

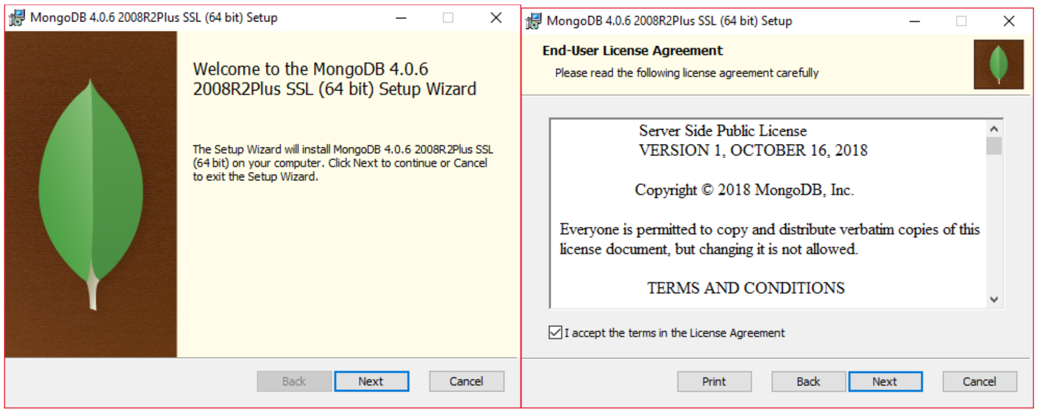
MongoDB Community Server
FEATURE RICH. DEVELOPER READY.

MongoDB Enterprise Server
ADVANCED FEATURES. PERFORMANCE GRADE.

Version: 4.0.6 (current release)
OS: Windows 64-bit x64
Package: MSI

https://fastdl.mongodb.org/win32/mongodb-win32-x86_64-2008plus-ssl-4.0.6-signed.msi

Once downloaded, proceed with the default installation process.


MongoDB 4.0.6 2008R2Plus SSL (64 bit) Setup

Welcome to the MongoDB 4.0.6 2008R2Plus SSL (64 bit) Setup Wizard

The Setup Wizard will install MongoDB 4.0.6 2008R2Plus SSL (64 bit) on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

MongoDB 4.0.6 2008R2Plus SSL (64 bit) Setup

End-User License Agreement
Please read the following license agreement carefully

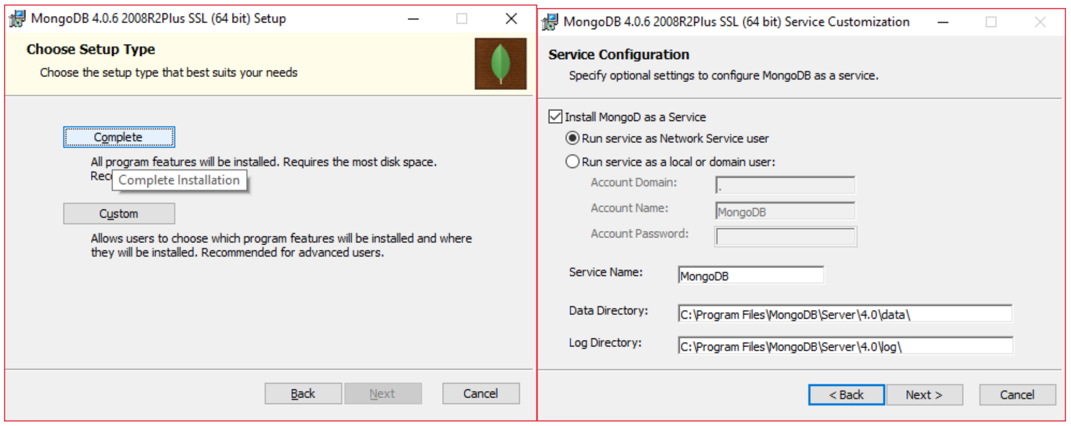
Server Side Public License
VERSION 1, OCTOBER 16, 2018

Copyright © 2018 MongoDB, Inc.

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

TERMS AND CONDITIONS

☑ I accept the terms in the License Agreement


MongoDB 4.0.6 2008R2Plus SSL (64 bit) Setup

Choose Setup Type
Choose the setup type that best suits your needs

Complete
All program features will be installed. Requires the most disk space.
Rec Complete Installation

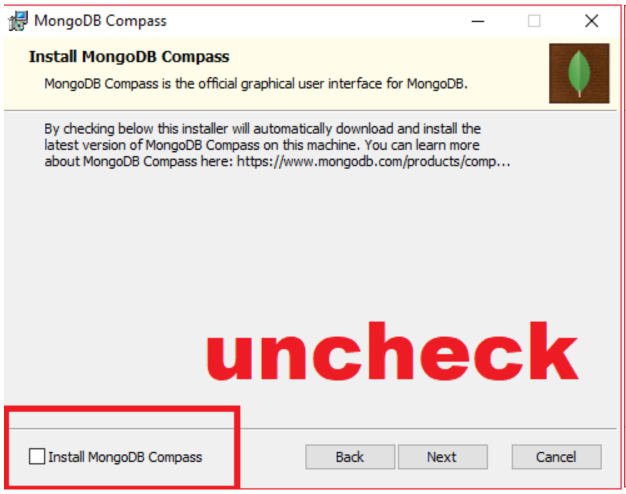
Custom
Allows users to choose which program features will be installed and where they will be installed. Recommended for advanced users.

MongoDB 4.0.6 2008R2Plus SSL (64 bit) Service Customization

Service Configuration
Specify optional settings to configure MongoDB as a service.

☑ Install MongoD as a Service
● Run service as Network Service user
○ Run service as a local or domain user:
Account Domain: .
Account Name: MongoDB
Account Password:

Service Name: MongoDB
Data Directory: C:\Program Files\MongoDB\Server\4.0\data\
Log Directory: C:\Program Files\MongoDB\Server\4.0\log\


MongoDB Compass

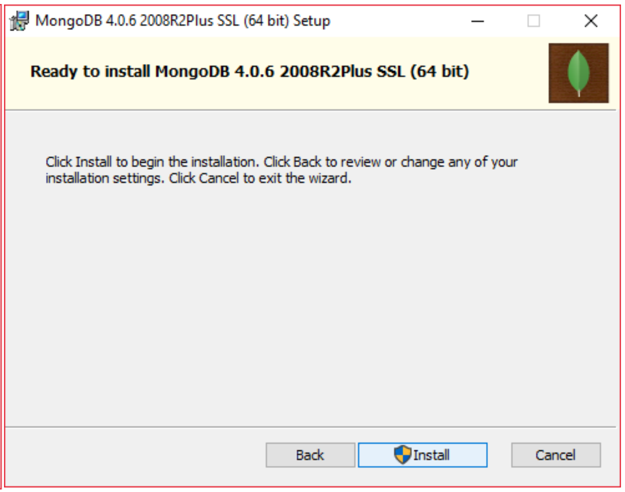
Install MongoDB Compass

MongoDB Compass is the official graphical user interface for MongoDB.

By checking below this installer will automatically download and install the latest version of MongoDB Compass on this machine. You can learn more about MongoDB Compass here: https://www.mongodb.com/products/comp...

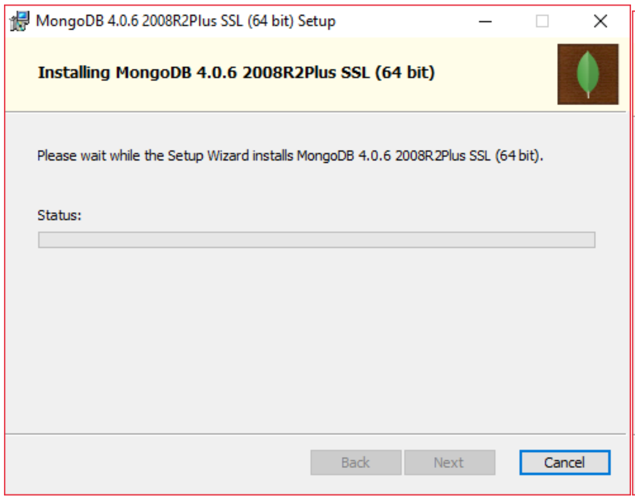
uncheck

☐ Install MongoDB Compass


MongoDB 4.0.6 2008R2Plus SSL (64 bit) Setup

Ready to install MongoDB 4.0.6 2008R2Plus SSL (64 bit)

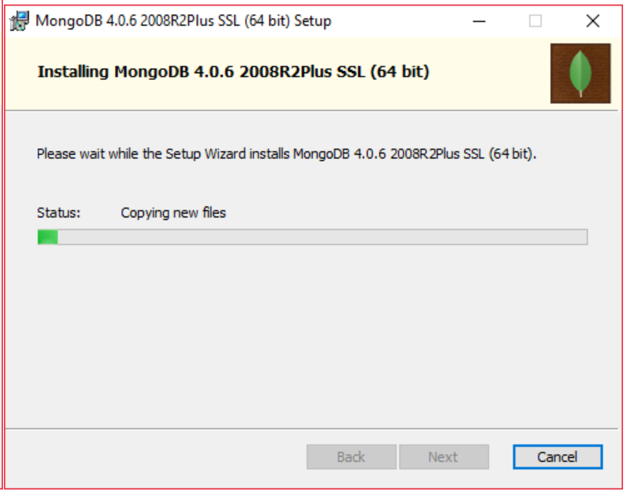
Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.


MongoDB 4.0.6 2008R2Plus SSL (64 bit) Setup

Installing MongoDB 4.0.6 2008R2Plus SSL (64 bit)

Please wait while the Setup Wizard installs MongoDB 4.0.6 2008R2Plus SSL (64 bit).

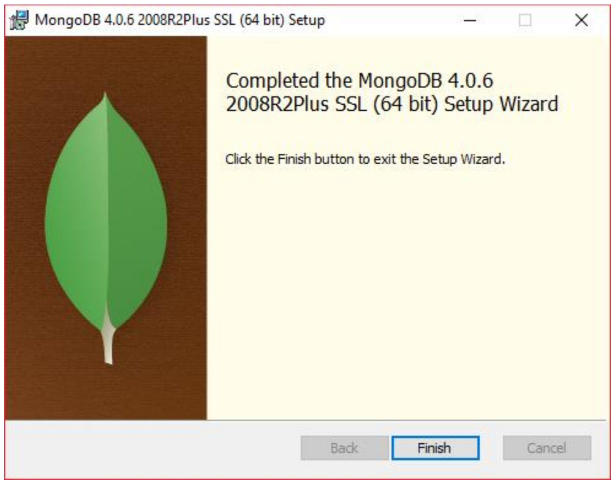
Status:


MongoDB 4.0.6 2008R2Plus SSL (64 bit) Setup

Installing MongoDB 4.0.6 2008R2Plus SSL (64 bit)

Please wait while the Setup Wizard installs MongoDB 4.0.6 2008R2Plus SSL (64 bit).

Status: Copying new files


MongoDB 4.0.6 2008R2Plus SSL (64 bit) Setup

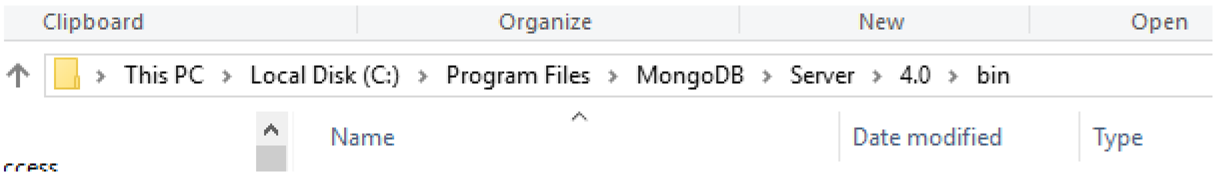
Completed the MongoDB 4.0.6 2008R2Plus SSL (64 bit) Setup Wizard

Click the Finish button to exit the Setup Wizard.

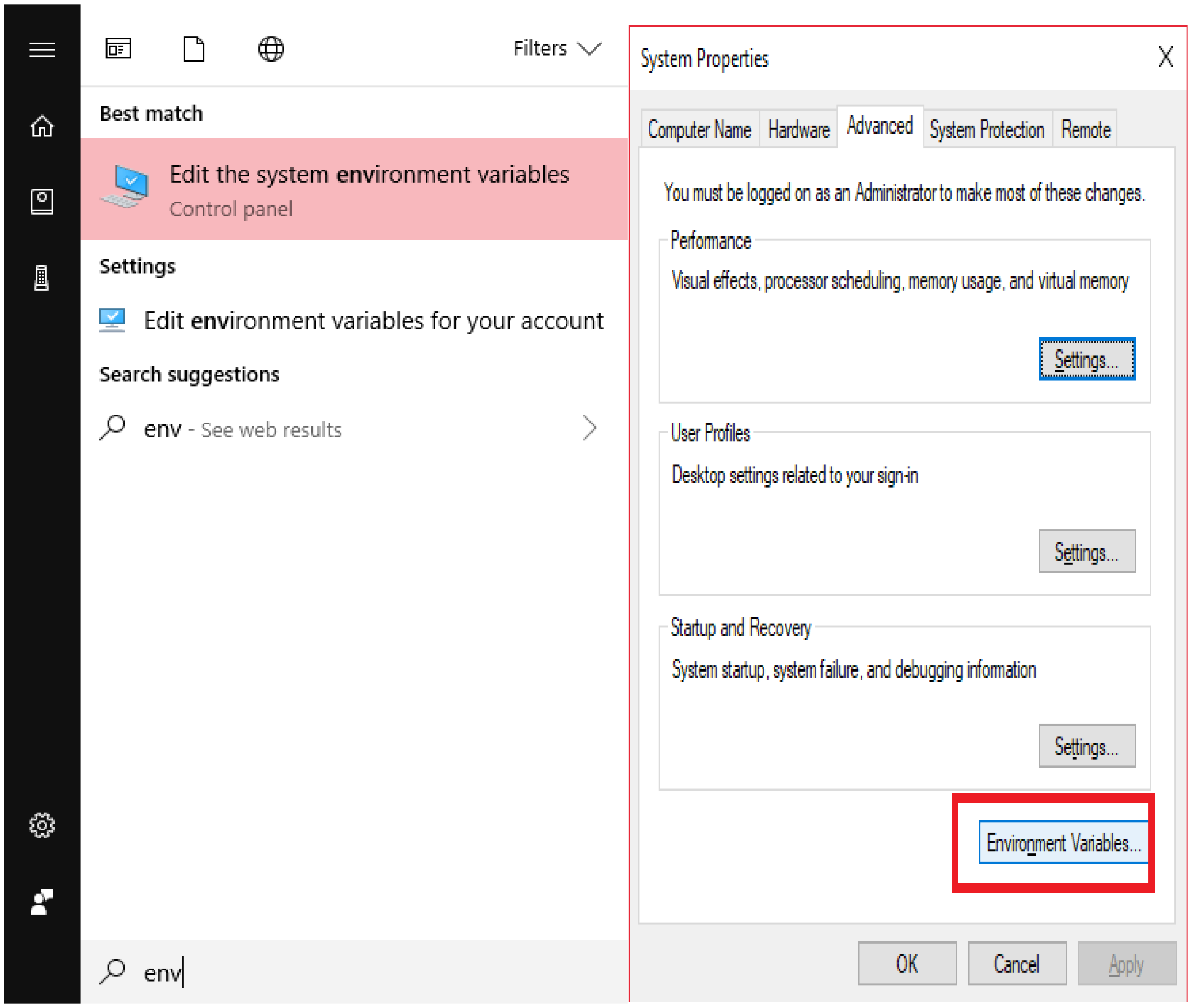
Once the Installation is done, You have to set the class path with the following steps.

Go to program files, and copy the path

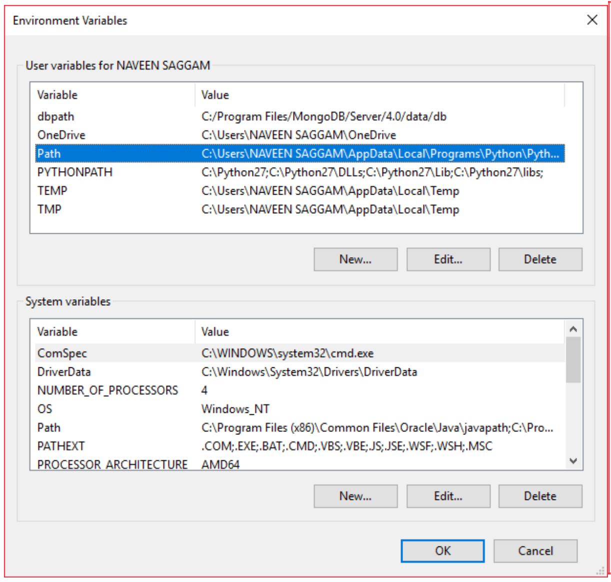
C:\Program Files\MongoDB\Server\4.0\bin


Clipboard
Organize
New
Open
This PC > Local Disk (C:) > Program Files > MongoDB > Server > 4.0 > bin
Name
Date modified
Type

Search for windows env variables using windows key. Go to Advanced Optoins, click on Env variables.


Filters
Best match
Edit the system environment variables
Control panel
Settings
Edit environment variables for your account
Search suggestions
env - See web results
env
System Properties
Computer Name
Hardware
Advanced
System Protection
Remote
You must be logged on as an Administrator to make most of these changes.
Performance
Visual effects, processor scheduling, memory usage, and virtual memory
Settings...
User Profiles
Desktop settings related to your sign-in
Settings...
Startup and Recovery
System startup, system failure, and debugging information
Settings...
Environment Variables...
OK
Cancel
Apply

Select the System Environment Variables, Select the Path and click on Edit and Click on New option, paste the Address you copied and click on OK => OK => OK.



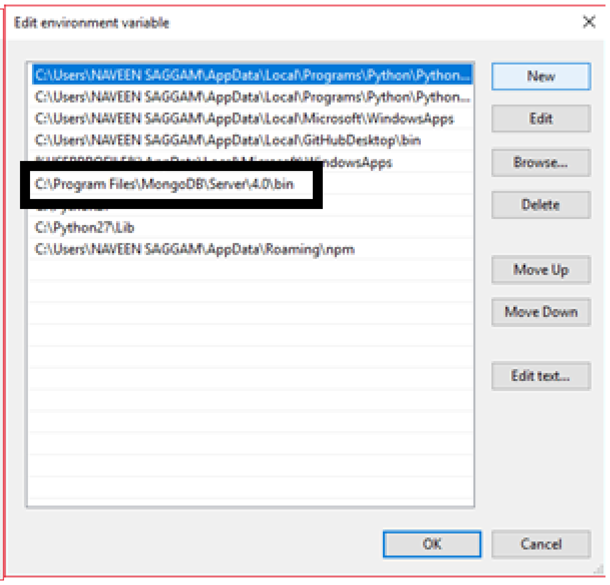
Environment Variables

User variables for NAVEEN SAGGAM

Variable	Value
dbpath	C:/Program Files/MongoDB/Server/4.0/data/db
OneDrive	C:\Users\NAVEEN SAGGAM\OneDrive
Path	C:\Users\NAVEEN SAGGAM\AppData\Local\Programs\Python\Pyth...
PYTHONPATH	C:\Python27;C:\Python27\DLLs;C:\Python27\Lib;C:\Python27\libs;
TEMP	C:\Users\NAVEEN SAGGAM\AppData\Local\Temp
TMP	C:\Users\NAVEEN SAGGAM\AppData\Local\Temp

System variables

Variable	Value
ComSpec	C:\WINDOWS\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
NUMBER_OF_PROCESSORS	4
OS	Windows_NT
Path	C:\Program Files (x86)\Common Files\Oracle\Java\javapath;C:\Pro...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE	AMD64

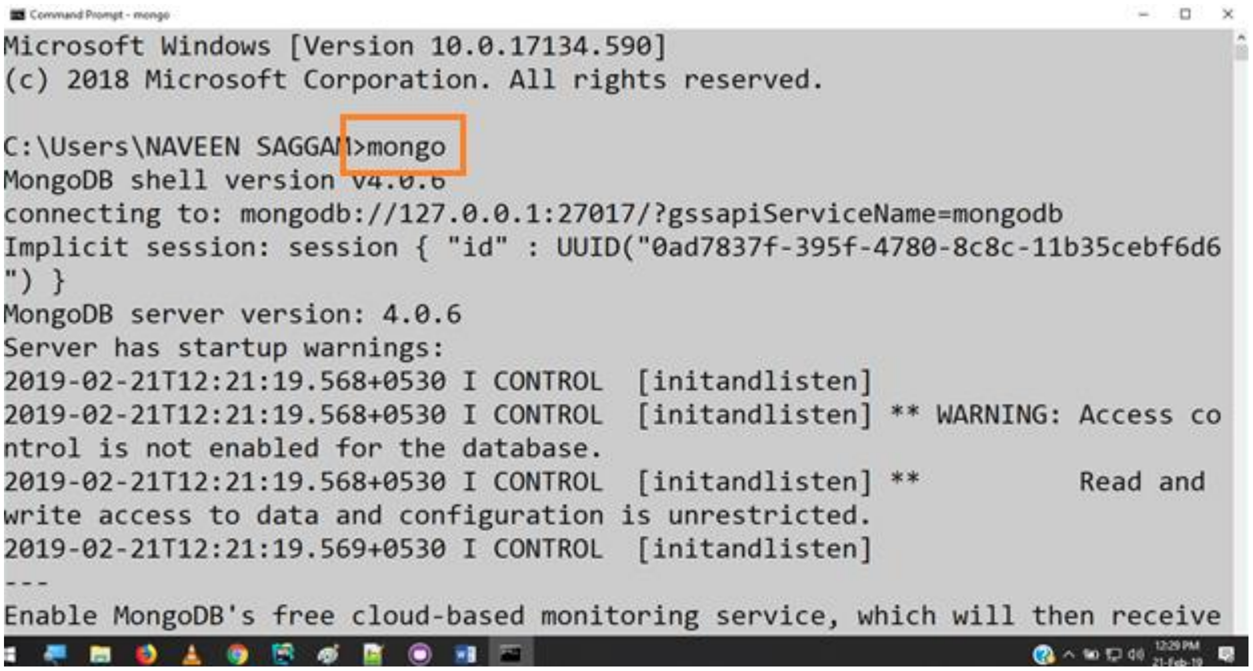


Edit environment variable

```
C:\Users\NAVEEN SAGGAM\AppData\Local\Programs\Python\Python...
C:\Users\NAVEEN SAGGAM\AppData\Local\Programs\Python\Python...
C:\Users\NAVEEN SAGGAM\AppData\Local\Microsoft\WindowsApps
C:\Users\NAVEEN SAGGAM\AppData\Local\GitHubDesktop\bin
%USERPROFILE%\AppData\Local\Microsoft\WindowsApps
C:\Program Files\MongoDB\Server\4.0\bin
[illegible]
C:\Python27\Lib
C:\Users\NAVEEN SAGGAM\AppData\Roaming\npm
```

Start MongoDB Shell

Go to Command Prompt and type the command **mongo**



Command Prompt - mongo

```
Microsoft Windows [Version 10.0.17134.590]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\NAVEEN SAGGAM>mongo
MongoDB shell version v4.0.6
connecting to: mongodb://127.0.0.1:27017/?gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("0ad7837f-395f-4780-8c8c-11b35cebf6d6
") }
MongoDB server version: 4.0.6
Server has startup warnings:
2019-02-21T12:21:19.568+0530 I CONTROL  [initandlisten]
2019-02-21T12:21:19.568+0530 I CONTROL  [initandlisten] ** WARNING: Access co
ntrol is not enabled for the database.
2019-02-21T12:21:19.568+0530 I CONTROL  [initandlisten] **          Read and
write access to data and configuration is unrestricted.
2019-02-21T12:21:19.569+0530 I CONTROL  [initandlisten]
---
Enable MongoDB's free cloud-based monitoring service, which will then receive
```

You should see the message like connecting to MongoDB, It says you have successfully installed the MongoDB and connected to it successfully.

MongoDB Create Database

In this we will see how to create a database in MongoDB

The use Command

MongoDB use DATABASE_NAME is used to create database. The command will create a new database if it doesn't exist, otherwise it will return the existing database.

Syntax

Basic syntax of use DATABASE statement is as follows

```
use DATABASE_NAME
```

If you want to use a database with name **showroom** then use DATABASE statement would be as follows

```
> use showroom
switched to db showroom
```

To check your currently selected database, use the command **db**

```
> db
showroom
```

If you want to check your databases list, use the command **show dbs**.

```
> show dbs
Act_Service_DB  0.000GB
admin           0.000GB
config          0.000GB
infosys_db      0.000GB
local           0.000GB
```

Your created database (showroom) is not present in list. To display database, you need to insert at least one document into it.

```
> db.showroom.insertOne({name:'Honda',location:'Hyderabad'});
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5c7d0cd36f4c910e9e80f63a")
}
```

```
> show dbs
Act_Service_DB  0.000GB
admin           0.000GB
config          0.000GB
infosys_db      0.000GB
local           0.000GB
showroom        0.000GB
```

Mongo DB Drop Database

MongoDB db.dropDatabase() command is used to drop a existing database.

To use this first your need to switch the required database and type this command to delete the database.

```
> use showroom
switched to db showroom
> db.dropDatabase();
{ "dropped" : "showroom", "ok" : 1 }
>
```

Now check the databases

```
> show databases
Act_Service_DB  0.000GB
admin           0.000GB
config          0.000GB
infosys_db      0.000GB
local           0.000GB
```

In the above list, **showroom** database is removed.

Mongo DB Create Collection

MongoDB db.createCollection(name, options) is used to create collection.

```
db.createCollection(name, options)
```

In the command, name is name of collection to be created. Options is a document and is used to specify configuration of collection.

Basic syntax of createCollection() method without options is as follows

```
> use showroom
switched to db showroom
> db.createCollection('employee');
{ "ok" : 1 }
>
```

You can check the created collection by using the command **show collections**.

```
> show collections
employee
```

In MongoDB, you don't need to create collection. MongoDB creates collection automatically, when you insert some document.

```
> db.employee.insertOne({name:'John',age : 40, desg: 'Manager'});
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5c7d13b16f4c910e9e80f63c")
}
```

```
> show collections
employee
```

Mongo DB Drop Collection

Mongo DB’s db.collection.drop() is used to drop a collection from the database.

Basic syntax of drop() command is as follows

```
db.COLLECTION_NAME.drop()
```

```
> use showroom
switched to db showroom
> show collections
employee
employee1
> db.employee1.drop();
true
> show collections
employee
```

Mongo DB Data types

MongoDB supports many data types. Some of them are

String – This is the most commonly used datatype to store the data. String in MongoDB must be UTF-8 valid.

Integer − This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server.

Boolean − This type is used to store a boolean (true/ false) value.

Double − This type is used to store floating point values.

Min/ **Max** keys − This type is used to compare a value against the lowest and highest BSON elements.

Arrays − This type is used to store arrays or list or multiple values into one key.

Timestamp − ctimestamp. This can be handy for recording when a document has been modified or added.

Object − This datatype is used for embedded documents.

Null − This type is used to store a Null value.

Symbol − This datatype is used identically to a string; however, it's generally reserved for languages that use a specific symbol type.

Date − This datatype is used to store the current date or time in UNIX time format. You can specify your own date time by creating object of Date and passing day, month, year into it.

Object ID − This datatype is used to store the document’s ID.

Binary data − This datatype is used to store binary data.

Code − This datatype is used to store JavaScript code into the document.

Regular expression − This datatype is used to store regular expression.

Mongo DB Create Operations

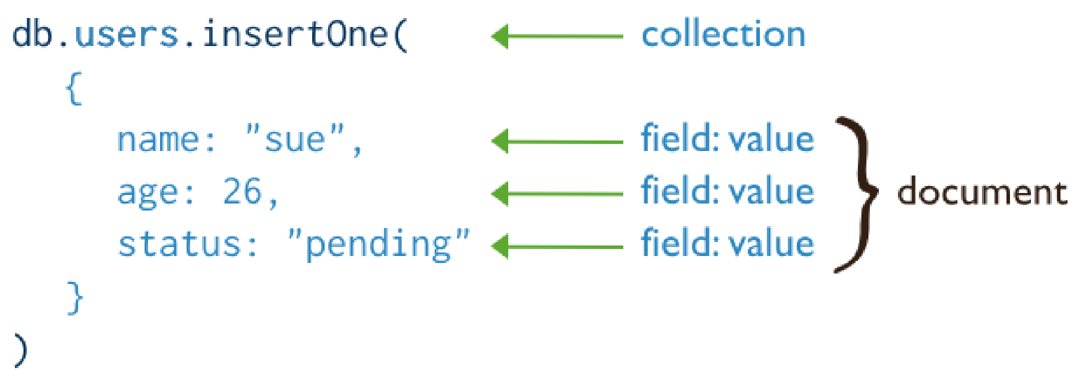
Create or insert operations add new documents to a collection. If the collection does not currently exist, insert operations will create the collection.

MongoDB provides the following methods to insert documents into a collection

db.collection.insertOne() New in version 3.2

db.collection.insertMany() New in version 3.2

In MongoDB, insert operations target a single collection. All write operations in MongoDB are atomic on the level of a single document.



```
db.users.insertOne(
   {
      name: "sue",
      age: 26,
      status: "pending"
   }
)
```

db.collection.insertOne() inserts a single document into a collection.

The following example inserts a new document into the inventory collection. If the document does not specify an _id field, MongoDB adds the _id field with an ObjectId value to the new document.

```
db.inventory.insertOne(
   { item: "canvas", qty: 100, tags: ["cotton"], size: { h: 28, w: 35.5, uom: "cm" } }
)
```

```
> use inventory
switched to db inventory
> db.inventory.insertOne(
...    { item: "canvas", qty: 100, tags: ["cotton"], size: { h: 28, w: 35.5, u
om: "cm" } }
... )
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5c7d15ce6f4c910e9e80f63d")
}
>
```

insertOne() returns a document that includes the newly inserted document's _id field value. For an example of a return document, see db.collection.insertOne() reference.

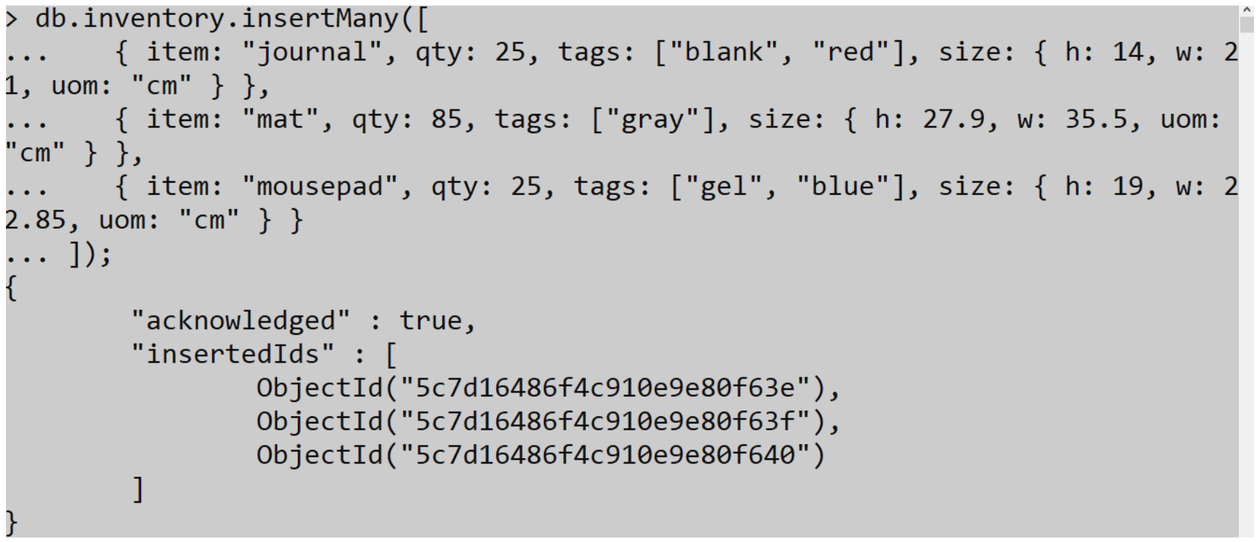
To retrieve the document that you just inserted, query the collection:

```
db.inventory.find( { item: "canvas" } )
```

db.collection.insertMany() can insert multiple documents into a collection. Pass an array of documents to the method.

The following example inserts three new documents into the inventory collection. If the documents do not specify an _id field, MongoDB adds the _id field with an ObjectId value to each document.

```
db.inventory.insertMany([
   { item: "journal", qty: 25, tags: ["blank", "red"], size: { h: 14, w: 21, uom: "cm" } },
   { item: "mat", qty: 85, tags: ["gray"], size: { h: 27.9, w: 35.5, uom: "cm" } },
   { item: "mousepad", qty: 25, tags: ["gel", "blue"], size: { h: 19, w: 22.85, uom: "cm" } }
])
```



```
> db.inventory.insertMany([
...    { item: "journal", qty: 25, tags: ["blank", "red"], size: { h: 14, w: 2
1, uom: "cm" } },
...    { item: "mat", qty: 85, tags: ["gray"], size: { h: 27.9, w: 35.5, uom:
"cm" } },
...    { item: "mousepad", qty: 25, tags: ["gel", "blue"], size: { h: 19, w: 2
2.85, uom: "cm" } }
... ]);
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("5c7d16486f4c910e9e80f63e"),
                ObjectId("5c7d16486f4c910e9e80f63f"),
                ObjectId("5c7d16486f4c910e9e80f640")
        ]
}
```

insertMany() returns a document that includes the newly inserted documents _id field values. See the reference for an example.

To retrieve the inserted documents, query the collection:

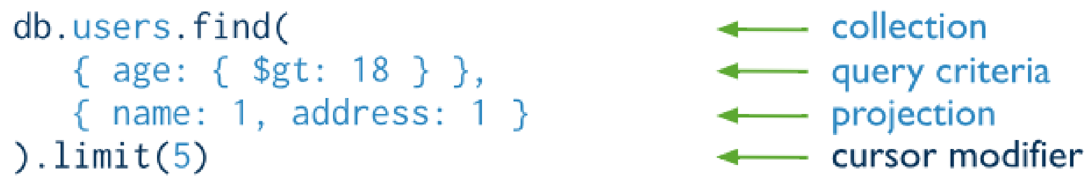
```
db.inventory.find( {} )
```

Mongo DB Read Operations

Read operations retrieves documents from a collection; i.e. queries a collection for documents. MongoDB provides the following methods to read documents from a collection:

```
db.collection.find()
```

You can specify query filters or criteria that identify the documents to return.



```
db.users.find(
   { age: { $gt: 18 } },
   { name: 1, address: 1 }
).limit(5)
```

To select all documents in the collection, pass an empty document as the query filter parameter to the find method. The query filter parameter determines the select criteria:

```
db.inventory.find( {} )
```

This operation corresponds to the following SQL statement:

```
SELECT * FROM inventory
```

The following example selects from the inventory collection all documents where the status equals "D":

```
// Mongo DB
db.inventory.find( { status: "D" } )

// SQL Query
SELECT * FROM inventory WHERE status = "D"
```

A query filter document can use the query operators to specify conditions in the following form

```
{ <field1>: { <operator1>: <value1> }, ... }
```

The following example retrieves all documents from the inventory collection where status equals either "A" or "D":

```
// Mongo DB
db.inventory.find( { status: { $in: [ "A", "D" ] } } )

// SQL Query
SELECT * FROM inventory WHERE status in ("A", "D")
```

A compound query can specify conditions for more than one field in the collection's documents. Implicitly, a logical AND conjunction connects the clauses of a compound query so that the query selects the documents in the collection that match all the conditions.

The following example retrieves all documents in the inventory collection where the status equals "A" and qty is less than ($lt) 30:

```
// Mongo DB
db.inventory.find( { status: "A", qty: { $lt: 30 } } )

// SQL Query
SELECT * FROM inventory WHERE status = "A" AND qty < 30
```

Using the $or operator, you can specify a compound query that joins each clause with a logical OR conjunction so that the query selects the documents in the collection that match at least one condition.

The following example retrieves all documents in the collection where the status equals "A" or qty is less than ($lt) 30:

```
// Mongo DB
db.inventory.find( { $or: [ { status: "A" }, { qty: { $lt: 30 } } ] } )

// SQL Query
SELECT * FROM inventory WHERE status = "A" OR qty < 30
```

In the following example, the compound query document selects all documents in the collection where the status equals "A" and either qty is less than ($lt) 30 or item starts with the character p:

```
// Mongo DB
db.inventory.find( {
     status: "A",
     $or: [ { qty: { $lt: 30 } }, { item: /^p/ } ]
} )

// SQL Query
SELECT * FROM inventory WHERE status = "A" AND ( qty < 30 OR item LIKE "p%")
```

Mongo DB Update Operations

The following methods are used to update any records / documents inside a collections.

```
db.collection.updateOne(<filter>, <update>, <options>)
db.collection.updateMany(<filter>, <update>, <options>)
db.collection.replaceOne(<filter>, <update>, <options>)
```

The following examples use the inventory collection. To create and/or populate the inventory collection

```
db.inventory.insertMany( [
   { item: "canvas", qty: 100, size: { h: 28, w: 35.5, uom: "cm" }, status: "A" },
   { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
   { item: "mat", qty: 85, size: { h: 27.9, w: 35.5, uom: "cm" }, status: "A" },
   { item: "mousepad", qty: 25, size: { h: 19, w: 22.85, uom: "cm" }, status: "P" },
   { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "P" },
   { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },
   { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },
   { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" },
   { item: "sketchbook", qty: 80, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
   { item: "sketch pad", qty: 95, size: { h: 22.85, w: 30.5, uom: "cm" }, status: "A" }
] );
```

To update a document, MongoDB provides update operators, such as $set, to modify field values.

To use the update operators, pass to the update methods an update document of the form:

```
{
  <update operator>: { <field1>: <value1>, ... },
  <update operator>: { <field2>: <value2>, ... },
  ...
}
```

Some update operators, such as $set, will create the field if the field does not exist. See the individual update operator reference for details.

The following example uses the db.collection.updateOne() method on the inventory collection to update the first document where item equals "paper":

```
db.inventory.updateOne(
   { item: "paper" },
   {
     $set: { "size.uom": "cm", status: "P" },
     $currentDate: { lastModified: true }
   }
)
```

The update operation:

uses the $set operator to update the value of the size.uom field to "cm" and the value of the status field to "P",

uses the $currentDate operator to update the value of the lastModified field to the current date. If lastModified field does not exist, $currentDate will create the field. See $currentDate for details.

The following example uses the db.collection.updateMany() method on the inventory collection to update all documents where qty is less than 50:

```
db.inventory.updateMany(
   { "qty": { $lt: 50 } },
   {
     $set: { "size.uom": "in", status: "P" },
     $currentDate: { lastModified: true }
   }
)
```

The update operation:

uses the $set operator to update the value of the size.uom field to "in" and the value of the status field to "P",

uses the $currentDate operator to update the value of the lastModified field to the current date. If lastModified field does not exist, $currentDate will create the field. See $currentDate for details.

Replace a Document

To replace the entire content of a document except for the _id field, pass an entirely new document as the second argument to **db.collection.replaceOne()**.

When replacing a document, the replacement document must consist of only field/value pairs; i.e. do not include update operators expressions.

The replacement document can have different fields from the original document. In the replacement document, you can omit the _id field since the _id field is immutable; however, if you do include the _id field, it must have the same value as the current value.

The following example replaces the first document from the inventory collection where item: "paper":

```
db.inventory.replaceOne(
   { item: "paper" },
   { item: "paper", instock: [ { warehouse: "A", qty: 60 }, { warehouse: "B", qty: 40 } ] }
)
```

Mongo DB Delete Operations

This Section uses the following mongo shell methods

```
db.collection.deleteMany()
db.collection.deleteOne()
```

The examples on this page use the inventory collection. To populate the inventory collection, run the following:

```
db.inventory.insertMany( [
   { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
   { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "P" },
   { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },
   { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },
   { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" },
] );
```

To delete all documents from a collection, pass an empty filter document {} to the **db.collection.deleteMany()** method.

The following example deletes all documents from the inventory collection

```
db.inventory.deleteMany({})
```

The method returns a document with the status of the operation. For more information and examples, see deleteMany().

You can specify criteria, or filters, that identify the documents to delete. The filters use the same syntax as read operations.

To specify equality conditions, use <field>:<value> expressions in the query filter document:

```
{ <field1>: <value1>, ... }
```

A query filter document can use the query operators to specify conditions in the following form:

```
{ <field1>: { <operator1>: <value1> }, ... }
```

To delete all documents that match a deletion criteria, pass a filter parameter to the **deleteMany()** method.

The following example removes all documents from the inventory collection where the status field equals "A":

```
db.inventory.deleteMany({ status : "A" })
```

The method returns a document with the status of the operation. For more information and examples, see **deleteMany().**

To delete at most a single document that matches a specified filter (even though multiple documents may match the specified filter) use the **db.collection.deleteOne()** method.

The following example deletes the first document where status is "D"

```
db.inventory.deleteOne( { status: "D" } )
```