www.BrainKart.com

Anna University

for Affilated Engineering College - 2021 Regulation

Information Technology

1st Semester | 2nd Semester | 3rd Semester | 4th Semester

5th Semester | 6th Semester | 7th Semester | 8th Semester

Click on Subject/Paper under Semester to enter.

1st Semester

- Professional English - I - HS3152
- Matrices and Calculus - MA3151
- Engineering Physics - PH3151
- Engineering Chemistry - CY3151
- Problem Solving and Python Programming - GE3151

2nd Semester

- Professional English - II - HS3252
- Statistics and Numerical Methods - MA3251
- Engineering Graphics - GE3251
- Physics for Information Science - PH3256
- Basic Electrical and Electronics Engineering - BE3251
- Programming in C - CS3251

3rd Semester

- Discrete Mathematics - MA3354
- Digital Principles and Computer Organization - CS3351
- Foundation of Data Science - CS3352
- Data Structures and Algorithms - CD3291
- Object Oriented Programming - CS3391

4th Semester

- Environmental Sciences and Sustainability - GE3451
- Theory of Computation - CS3452
- Artificial Intelligence and Machine Learning - CS3491
- Database Management System - CS3492
- Web Essentials - IT3401
- Introduction to Operating Systems - CS3451

5th Semester

- Computer Networks - CS3591
- Full Stack Web Development - IT3501
- Distributed Computing - CS3551
- Embedded Systems and IoT - CS3691
- Elective 1
- Elective 2

6th Semester

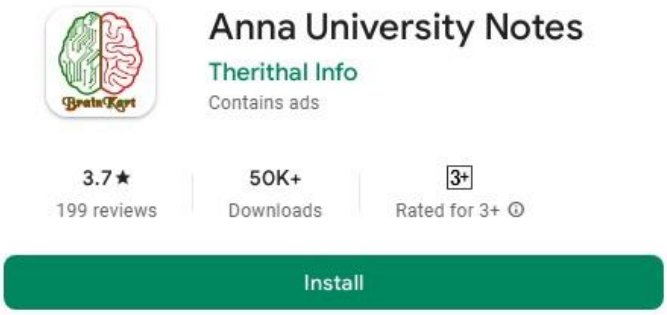
- Object Oriented Software Engineering - CCS356
- Open Elective-1
- Elective-3
- Elective-4
- Elective-5
- Elective-6

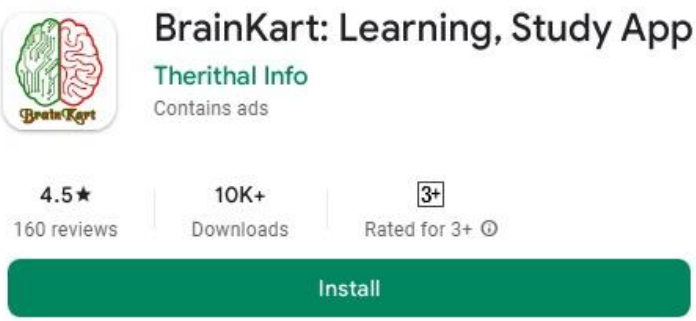
7th Semester

- Human Values and Ethics - GE3791
- Open Elective 2
- Open Elective 3
- Open Elective 4
- Management Elective

8th Semester

- Project Work / Intership





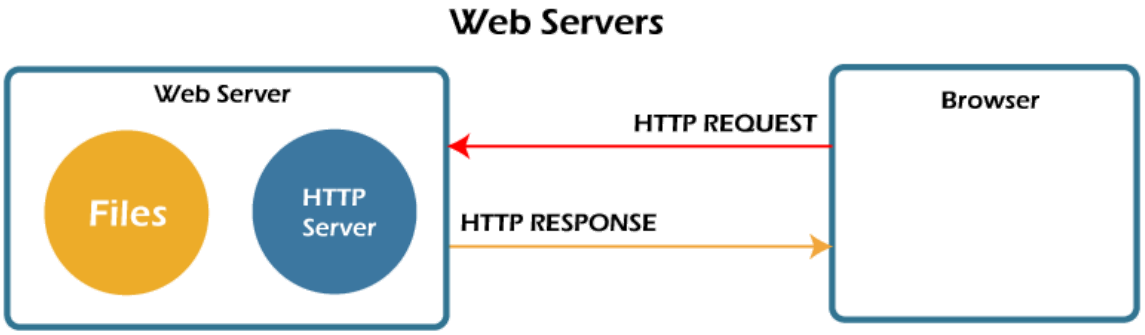
All Computer Engg Subjects - [B.E., M.E.,]		(Click on Subjects to enter)
Programming in C	Computer Networks	Operating Systems
Programming and Data Structures I	Programming and Data Structure II	Problem Solving and Python Programming
Database Management Systems	Computer Architecture	Analog and Digital Communication
Design and Analysis of Algorithms	Microprocessors and Microcontrollers	Object Oriented Analysis and Design
Software Engineering	Discrete Mathematics	Internet Programming
Theory of Computation	Computer Graphics	Distributed Systems
Mobile Computing	Compiler Design	Digital Signal Processing
Artificial Intelligence	Software Testing	Grid and Cloud Computing
Data Ware Housing and Data Mining	Cryptography and Network Security	Resource Management Techniques
Service Oriented Architecture	Embedded and Real Time Systems	Multi - Core Architectures and Programming
Probability and Queueing Theory	Physics for Information Science	Transforms and Partial Differential Equations
Technical English	Engineering Physics	Engineering Chemistry
Engineering Graphics	Total Quality Management	Professional Ethics in Engineering
Basic Electrical and Electronics and Measurement Engineering	Problem Solving and Python Programming	Environmental Science and Engineering

FULL STACK WEB DEVELOPMENT –

UNIT 2

SERVER SIDE PROGRAMMING WITH NODE JS

Web Servers



Web pages are a collection of data, including images, text files, hyperlinks, database files etc., all located on some computer (also known as server space) on the Internet. A web server is dedicated software that runs on the server-side. When any user requests their web browser to run any web page, the webserver places all the data materials together into an organized web page and forwards them back to the web browser with the help of the Internet. Therefore, we can conclude that: -

A web server is a dedicated computer responsible for running websites sitting out on those computers somewhere on the Internet. They are specialized programs that circulate web pages as summoned by the user. The primary objective of any web server is to collect, process and provide web pages to the users.

This intercommunication of a web server with a web browser is done with the help of a protocol named **HTTP (Hypertext Transfer Protocol).**

These stored web pages mostly use static content, containing HTML

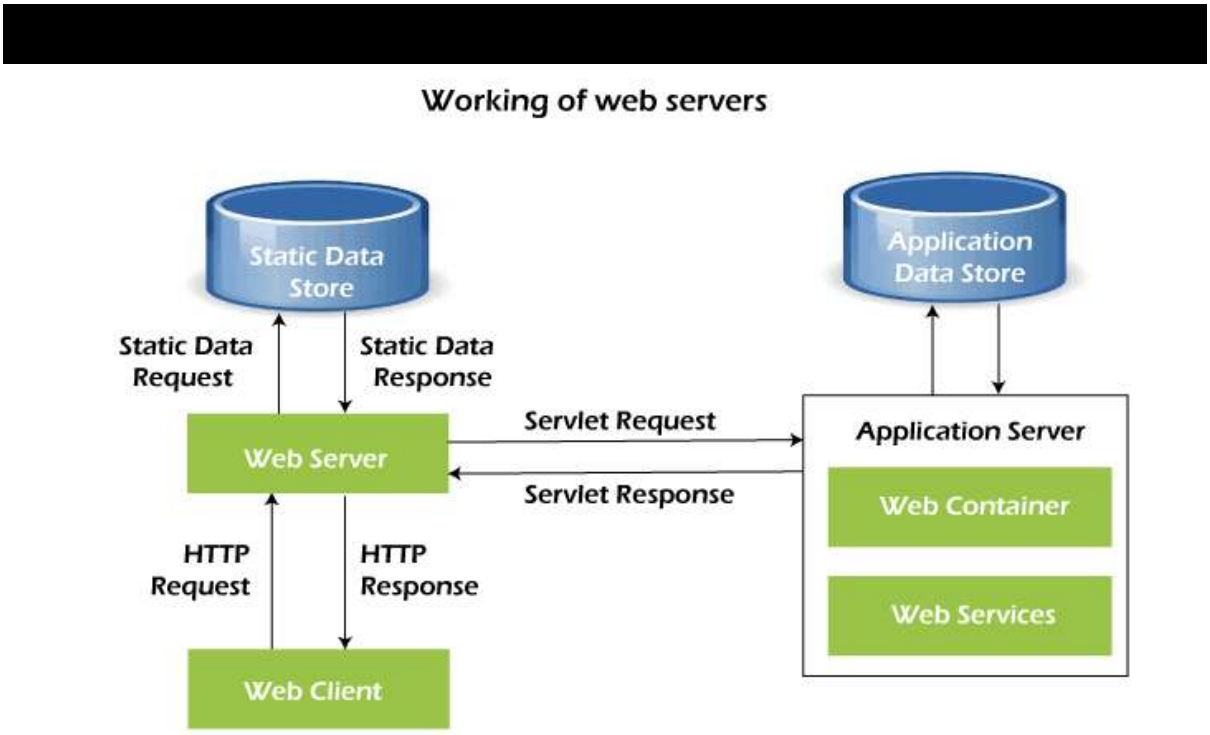
documents, images, style sheets, text files, etc. However, **web servers can serve static as well as dynamic contents**. Web Servers also assists in emailing services and storing files. Therefore it also uses SMTP (Simple Mail Transfer Protocol) and FTP (File Transfer Protocol)

protocols to support the respective services. Web servers are mainly used in web hosting or hosting the website's data and running web-based applications.

The hardware of the web servers are connected to the Internet that manages the data exchange facility within different connected devices. In contrast, the software of web server software is responsible for controlling how a user accesses delivered files. Typically, web server management is an ideal example of the client/server model. Therefore, **it is compulsory for all computers that host websites (whether with state or dynamic web page content) to have web server software.**

How do web servers work?

The term web server can denote server hardware or server software, or in most cases, both hardware and software



1. **On the hardware side**, a web server is defined as a computer that stores software and another website raw data, such as HTML files, images, text documents, and JavaScript files. The hardware of the web servers are connected to the web and supports the data exchange with different devices connected to the Internet.
2. **On the software side**, a web server includes server software accessed through website domain names. It controls how web users access the web files and ensures the supply of website content to the end-user. The web server contains several components, including an HTTP server.

Whenever any web browser

, such as Google Chrome

, Microsoft Edge

or Firefox

, requests for a web page hosted on a web server, the browser will process the request forward with the help of HTTP. At the server end, when it receives the request, the HTTP

server will accept the request and immediately start looking for the requested data and forwards it back to the web browser via HTTP.

Let's discover the step-by-step process of what happens whenever a web browser approaches the web server and requests a web file or file. Follow the below steps:

1. First, any web user is required to **type the URL of the web page in the address bar** of your web browser.
2. With the help of the URL, your **web browser will fetch the IP address of your domain** name either by converting the URL via DNS (Domain Name System) or by looking for the IP in cache memory. The IP address will direct your browser to the web server.
3. After making the connection, the **web browser will request for the web page from the web server** with the help of an HTTP request.
4. As soon as the web server receives this request, it immediately **responds by sending back the requested page** or file to the web browser HTTP.
5. If the web page requested by the **browser does not exist or if there occurs some error in the process**, the web server will return an error message.
6. If there occurs no error, the browser will successfully display the webpage.

Examples of web server uses

Web servers are mostly used for:

- sending and receiving mails on Internet by using SMTP (Simple Mail transfer Protocol);
- fetching requests for File Transfer Protocol (FTP) files; and
- designing, developing, and publishing websites.

Many Web servers, even the basic one, also support the server-side scripting technique. Server-side scripting is a web development method used to employ scripts on a web server that produces a customized response for each user. This technique operates on the server machine and consists of an extensive feature set, including database access. The server-side scripting process will have various scripting languages such ASP

, PHP

, Java

, JavaScript

, Python

, ruby

and many more. This technique also enables the HTML files to be created dynamically.

Static web servers vs. Dynamic web servers

A web server can be used to store, process or deliver either static or dynamic web pages. Let's understands the definition of static and dynamic web servers with the help of their difference table given below:

S.NO.	Static Web Servers	Dynamic Web Servers

1	Static web servers refer to the servers, which serve only the static content i.e., the content is fixed and being shown as it is.	Dynamic web servers refer to the servers where the content of the page can be updated and altered.
2	A static web server includes a computer and the HTTP (Hyper Text Transfer Protocol) software.	A dynamic web server also includes a computer with plenty of other software, unlike an application server and database model.
3	It is called static; the web pages content won't change unless the user manually changes it, and the server will deliver web files as is to the web browser.	It is called dynamic because the application server is used to update the web pages files at the server-side, and due to which, it can change on every call requested by the web browser.
4	Static web servers take less time to load the data.	The Dynamic web server can only produce the data when it is requested from the database. Therefore, it is time consuming and more complicated when compared to static web servers.

Web server software available in the market

Though there are various web servers found in today's market, but the commonly used one are as follows:

S.NO	Web Server Description
1.	**Apache HTTP Server** This web server is developed by Apache Software Foundation. It is an open-source, accessible web server available for almost all operating systems, including Windows, MACOS, Linus, FreeBSD, etc. Apache is one of the most popular web servers used around the globe.
2.	**Microsoft Internet Information Services (IIS)** IIS is a high-performance web server that is developed by Microsoft only for Microsoft platforms. This webs server is tightly integrated with Microsoft operating system; therefore, it is not open-sourced.
3.	**Nginx** Nginx is an open-source web server commonly used by administrators as it supports light resource application and scalability.

4.	**Lighttpd** Lighttpd, also known as lighty, is a free, open-source web server with the FreeBSD operating system. This web server is fast, secure and consumes much less CPU power. It can also run on the commonly used operating system, unlike Windows, Mac OS X, Linus.
5.	**Sun Java System Web Server** Sun Java is a free web server developed by Sun Microsystems well equipped for a medium and large website that can run on Windows, Linux and Unix. Moreover, this web server supports several languages, scripts and technologies essential for Web 2.0, unlike JSP, Java Servlets, PHP, Python, HTML, etc. Though Sun Java is free, it is not an open-source web server.

Web server security Methods

Though there are various security techniques available in the market that a user can implement to have a safe web server experience, below given are some examples of security practices that can include processes:

1. **A reverse proxy** is a proxy server that is accessible to the clients, therefore hiding the internal server. It acts as an intermediary as wherever any user makes requests to the web server for any data or file, the proxy server seizes those requests and then communicates with the webserver.
2. **Access restriction** is a technique that limits the web host's access to infrastructure machines or using Secure Socket Shell (SSH);
3. **Keeping web servers mended and updated**, as it benefits to ensure the web server isn't vulnerable to exposures;
4. **Network monitoring** is a security practice that ensures that no unauthorized activity is going on the web server; and
5. **Using a firewall and SSL safeguards** the web server as firewalls can supervise HTTP request traffic while a Secure Sockets Layer (SSL) supports securing the data.

JAVASCRIPT IN THE DESKTOP WITH NODEJS

Node.js Tutorial

Node.js tutorial provides basic and advanced concepts of Node.js. Our Node.js tutorial is designed for beginners and professionals both.

Node.js is a cross-platform environment and library for running JavaScript applications which is used to create networking and server-side applications.

Our Node.js tutorial includes all topics of Node.js such as Node.js installation on windows and linux, REPL, package manager, callbacks, event loop, os, path, query string, cryptography, debugger, URL, DNS, Net, UDP, process, child processes, buffers, streams, file

systems, global objects, web modules etc. There are also given Node.js interview questions to help you better understand the Node.js technology.

What is Node.js

Node.js is a cross-platform runtime environment and library for running JavaScript applications outside the browser. It is used for creating server-side and networking web applications. It is open source and free to use.

Many of the basic modules of Node.js are written in JavaScript. Node.js is mostly used to run real-time server applications.

The definition given by its official documentation is as follows:

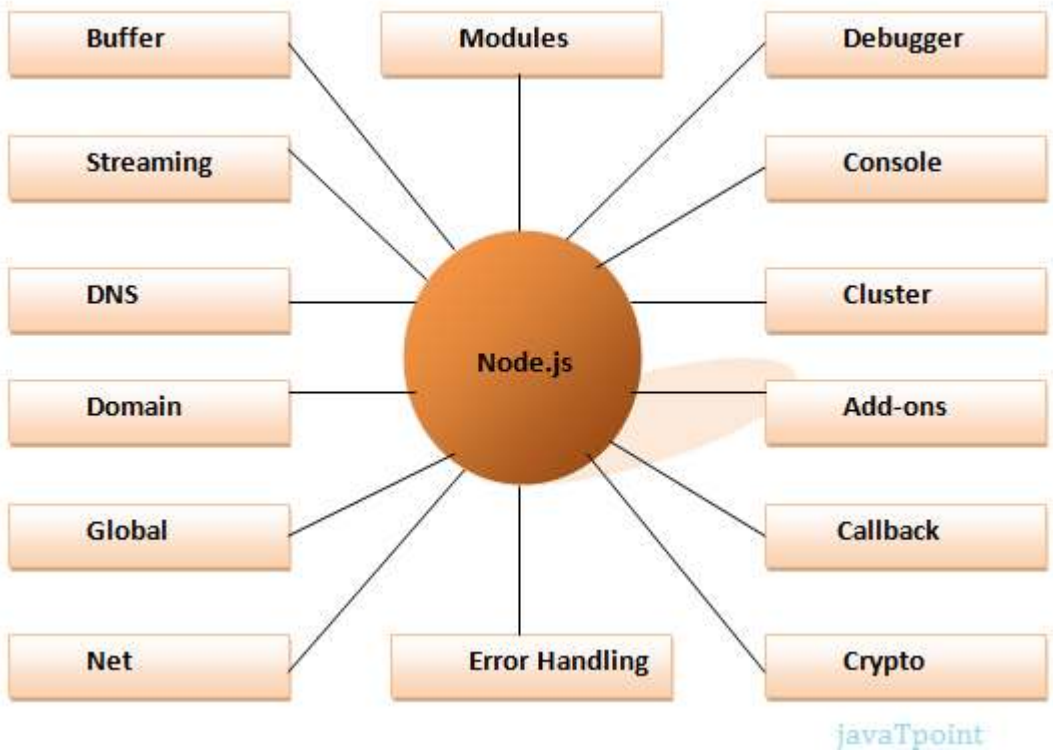
?Node.js is a platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.?

Node.js also provides a rich library of various JavaScript modules to simplify the development of web applications.

1. Node.js = Runtime Environment + JavaScript Library

Different parts of Node.js

The following diagram specifies some important parts of Node.js:



Features of Node.js

Following is a list of some important features of Node.js that makes it the first choice of software architects.

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

NPM

What is npm

npm is a short form of **Node Package Manager,** which is the world's largest software registry. The open-source web project developers use it from the entire world to **share** and **borrow** packages. The npm also acts as a command-line utility for the Node.js project for installing packages in the project, dependency management, and even version management.

Components of npm

npm mainly consists of three different components, these are:

1. **Website:** The npm official website is used to find packages for your project, create and set up profiles to manage and access private and public packages.
2. **Command Line Interface (CLI):** The CLI runs from your computer's terminal to interact with npm packages and repositories.
3. **Registry:** The registry is a large and public database of JavaScript projects and meta-information. You can use any supported npm registry that you want or even your own. You can even use someone else's registry as per their terms of use.

Installing npm (Node.js and npm)

npm comes with **Node.js,** which means you have to install Node.js to get installed automatically on your personal computer. There are two different methods to install npm on a computer.

1. **Using nvm (Node Version Manager)**
2. **Using Node installer**

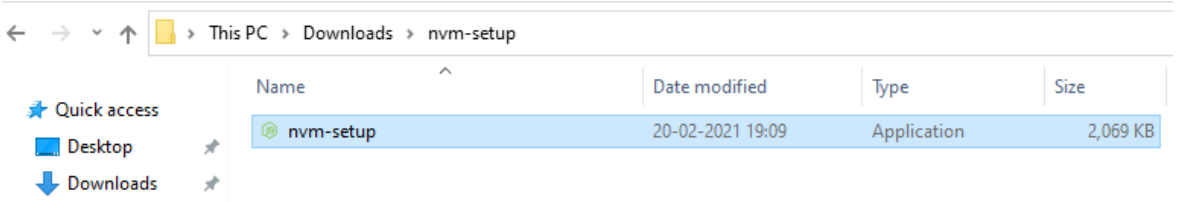
npm installation is highly recommended using nvm (Node Version Manager) rather than using Node installer. The Node installation process installs npm in the directory by allowing local permissions, and it generates an error message when you try to run npm packages globally.

1. Install npm and Node.js using nvm

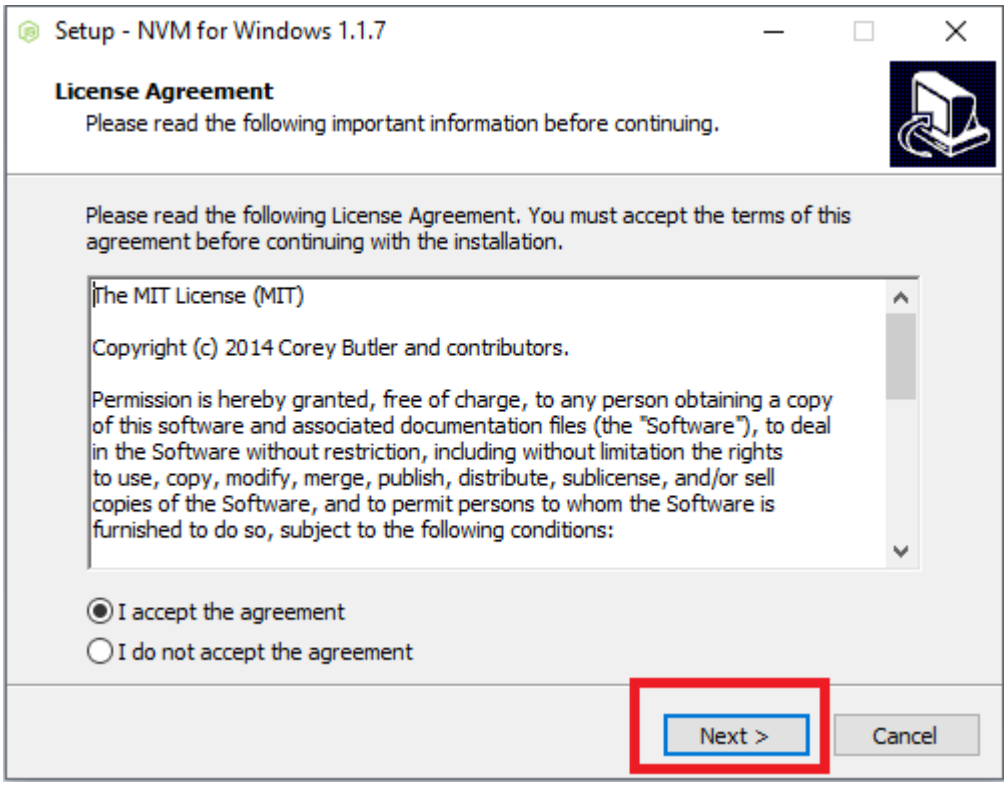
It is highly recommended by npm's to use Node version manager to install npm and Node.js on your device. Installing node.js using nvm allows installing different versions of Node.js on your computer, and you can use any Node.js version for your project by switching as per your requirement. They do not recommend to us Node installers to install npm and Node.js.

To download and install npm using nvm on your **Linux operating system,** open https://github.com/nvm-sh/nvm.

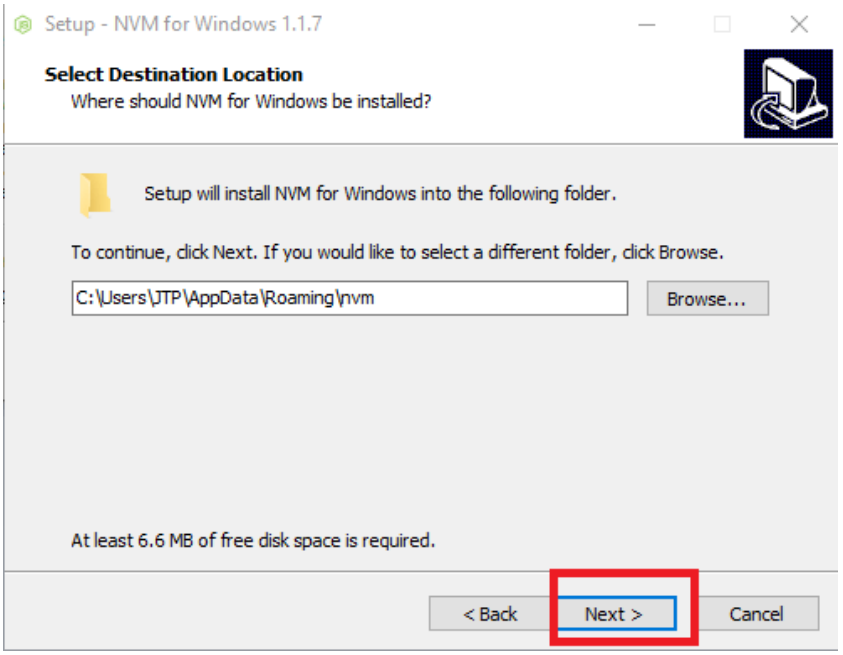
1. We will download the **nvm-setup.zip** file for Windows from Git link as we are using a Windows OS device.
2. After downloading, run the setup and follow the installation instruction as appears on the screen.



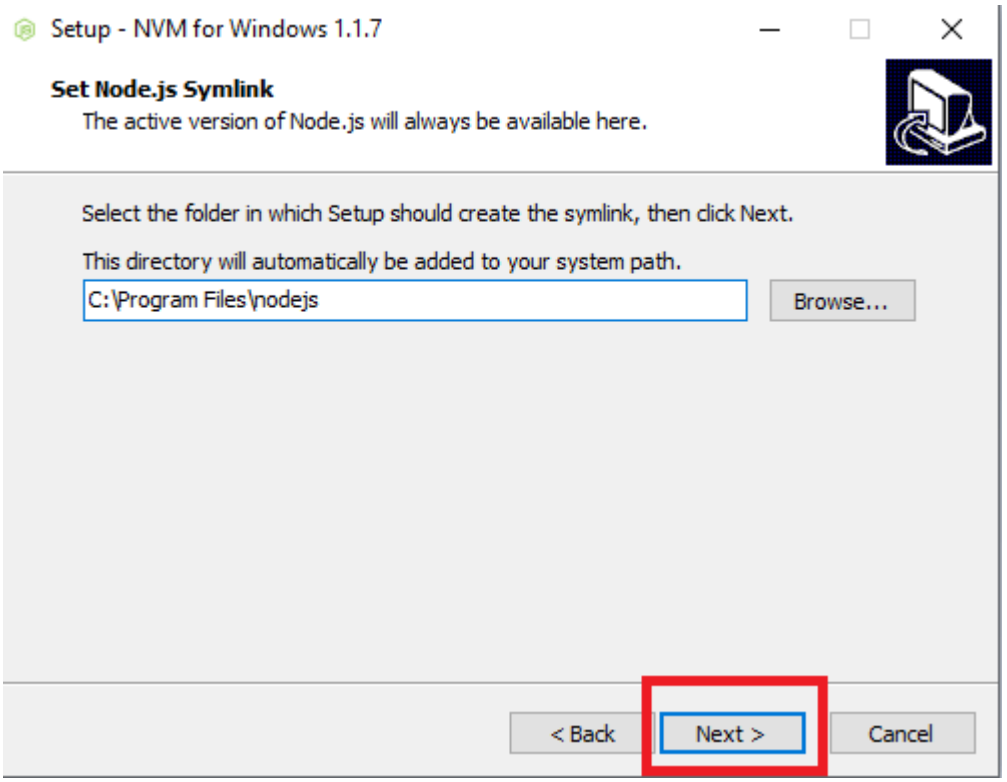
3. Run the setup either by double-clicking on it or by right-clicking and Run as administration.
4. Allow the installation by clicking on the **Yes** button.
5. Read their license agreement carefully and if you agree with it, then accept it and click on **Next** button.



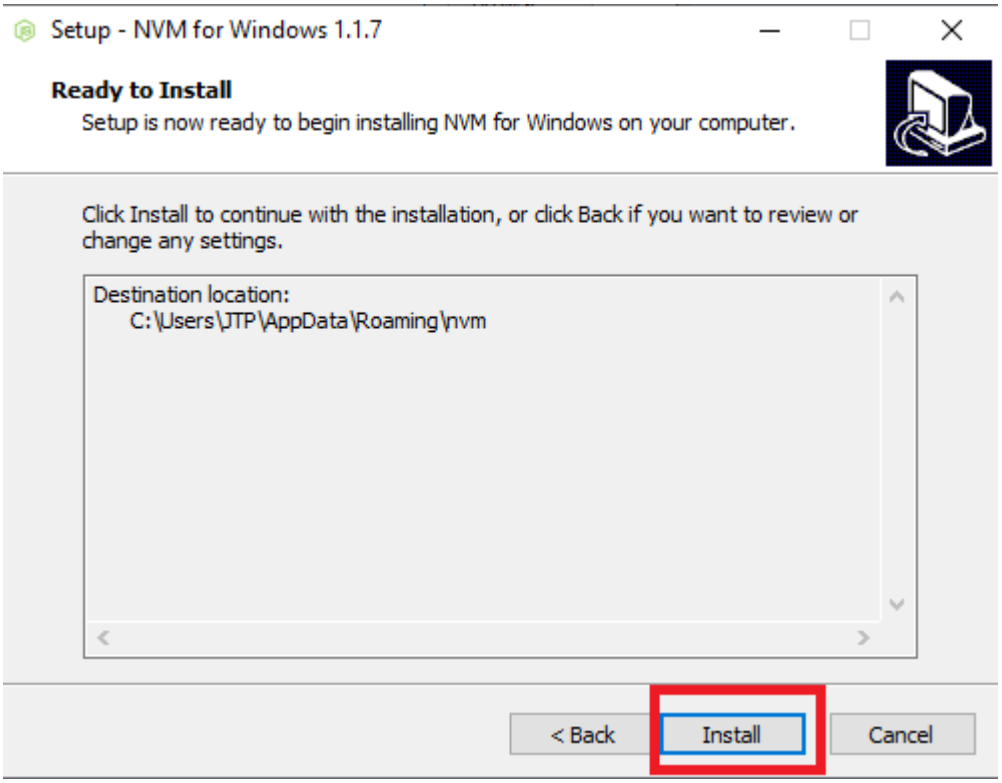
6. Select the destination location where you want to install NVM and click on the **Next** button. We leave it as the default selected location.



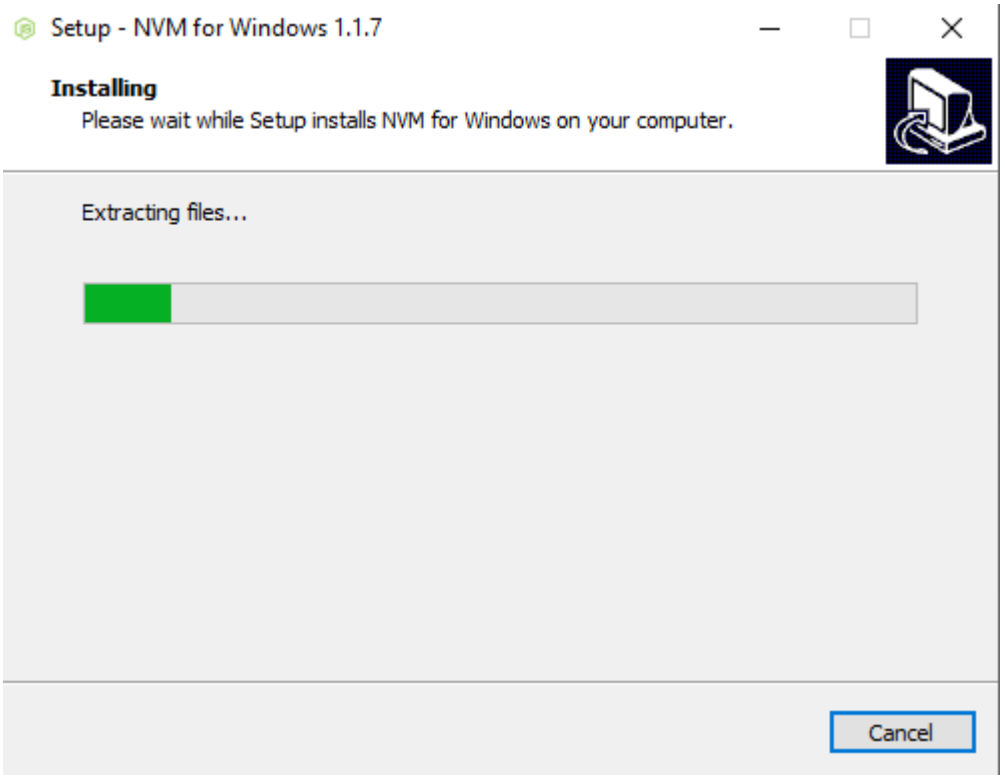
7. Select the folder location in which the setup will create the symlink and click on the **Next** button.



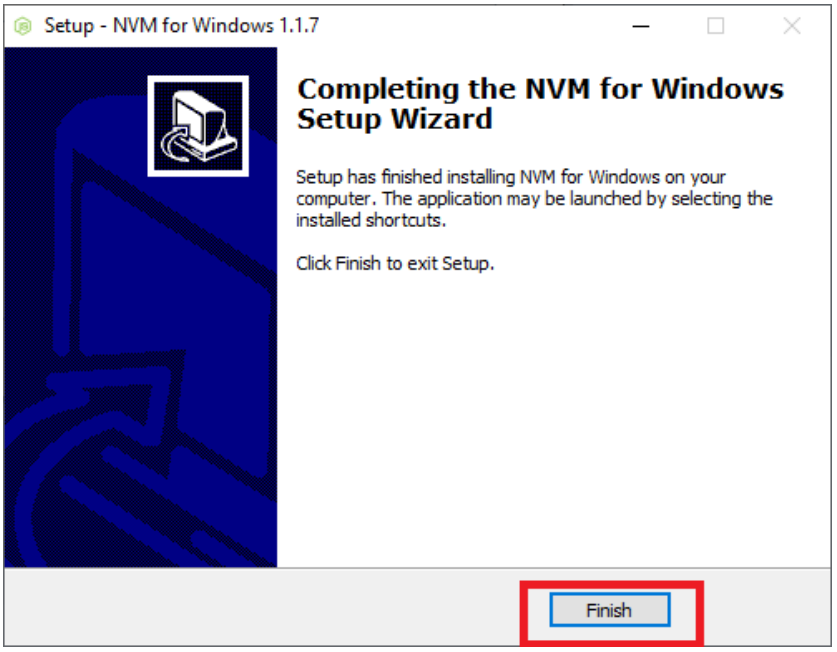
8. Now, your setup is ready to install, click on the **Install** button to start the installation process.



9. Now, the nvm-setup is installed on your personal computer, and it will take few seconds to wait to complete it.



10. Your nvm setup installation gets finished; click on the **Finish** button to exit from setup.

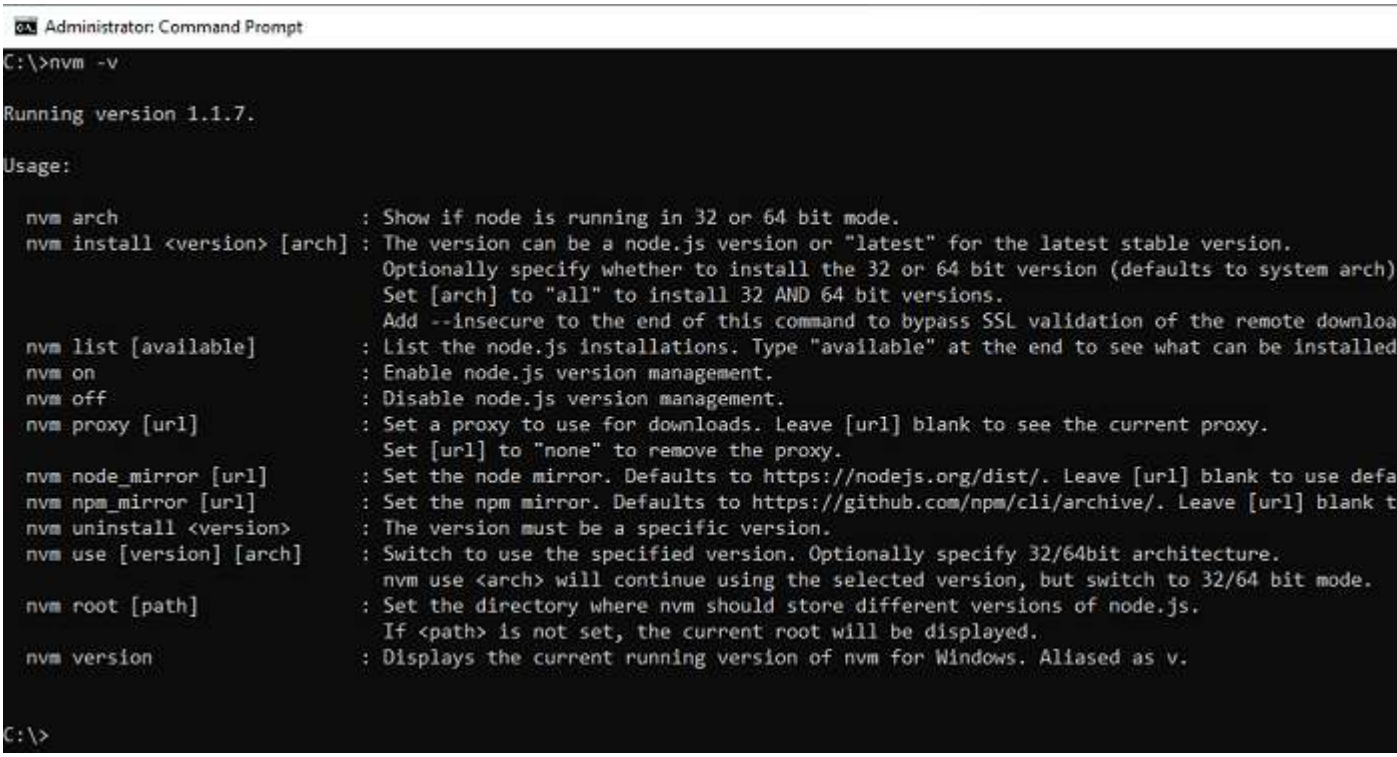


Check nvm version

After the nvm setup gets installed, ensure whether you get installed or which version on nvm is installed. To do this, run the following command on your terminal:

nvm - v

The above command displays the nvm version running on your device and another nvm command with their short descriptions.



You can also check the nvm version using the command: **nvm version.**

Check the list of Node.js available to install using nvm

To find the supported Node.js version to install using nvm, use the following command. It will display the partial available list of Node.js versions.

1. npm list available

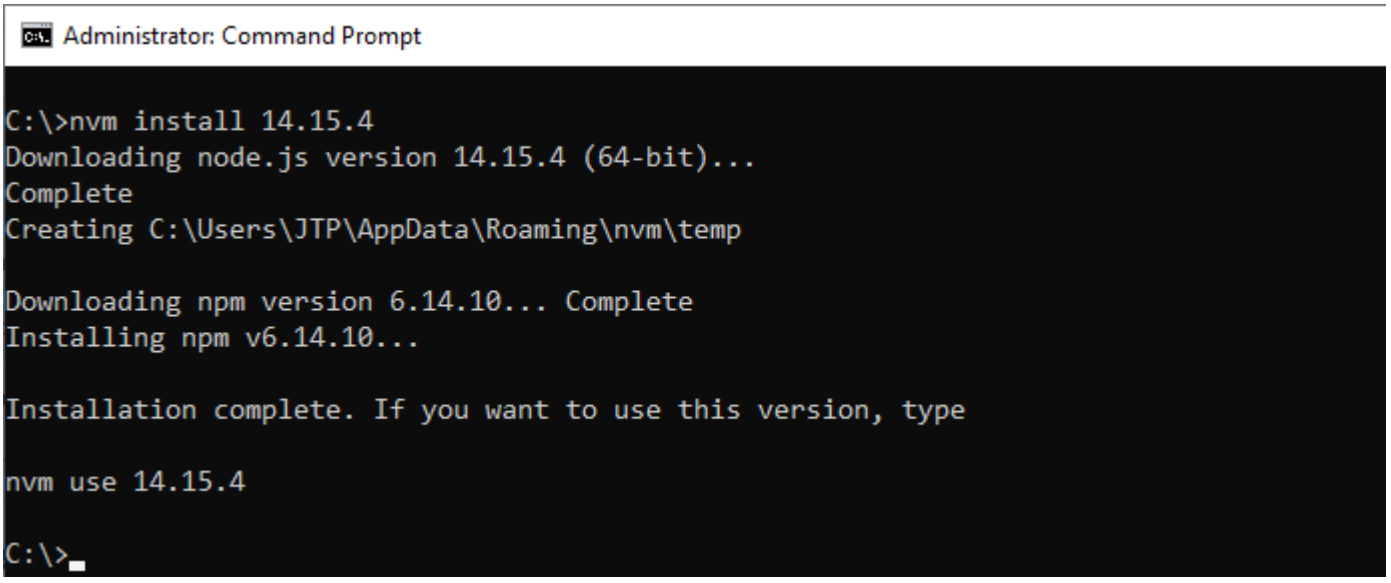
```
C:\>nvm list available

|   CURRENT    |     LTS      | OLD STABLE  | OLD UNSTABLE |
|--------------|--------------|-------------|--------------|
|    15.9.0    |   14.15.5    |   0.12.18   |   0.11.16    |
|    15.8.0    |   14.15.4    |   0.12.17   |   0.11.15    |
|    15.7.0    |   14.15.3    |   0.12.16   |   0.11.14    |
|    15.6.0    |   14.15.2    |   0.12.15   |   0.11.13    |
|    15.5.1    |   14.15.1    |   0.12.14   |   0.11.12    |
|    15.5.0    |   14.15.0    |   0.12.13   |   0.11.11    |
|    15.4.0    |   12.20.2    |   0.12.12   |   0.11.10    |
|    15.3.0    |   12.20.1    |   0.12.11   |    0.11.9    |
|    15.2.1    |   12.20.0    |   0.12.10   |    0.11.8    |
|    15.2.0    |   12.19.1    |    0.12.9   |    0.11.7    |
|    15.1.0    |   12.19.0    |    0.12.8   |    0.11.6    |
|    15.0.1    |   12.18.4    |    0.12.7   |    0.11.5    |
|    15.0.0    |   12.18.3    |    0.12.6   |    0.11.4    |
|   14.14.0    |   12.18.2    |    0.12.5   |    0.11.3    |
|   14.13.1    |   12.18.1    |    0.12.4   |    0.11.2    |
|   14.13.0    |   12.18.0    |    0.12.3   |    0.11.1    |
|   14.12.0    |   12.17.0    |    0.12.2   |    0.11.0    |
|   14.11.0    |   12.16.3    |    0.12.1   |    0.9.12    |
|   14.10.1    |   12.16.2    |    0.12.0   |    0.9.11    |
|   14.10.0    |   12.16.1    |   0.10.48   |    0.9.10    |

This is a partial list. For a complete list, visit https://nodejs.org/download/release
```

In the above list, **CURRENT** refers to the current version of Node.js, and **LTS** refers to the long-term support version.

Installing Node.js and npm

You can choose any version of Node.js to install; we are installing LTS 14.15.4 for our machine. Simply run the following command to install Node.js:

1. nvm install 14.15.4



This command installs Node.js (64-bit or 32-bit) and npm together depending on your device.

Check the version of Node.js and npm

To use Node.js use the command **nvm use 14.15.4** (make sure to replace 14.15.4 with your installed version).

1. nvm use 14.15.4

After that, run the command **node -v** and **npm -v** on your terminal to check the version of Node.js and npm, respectively.

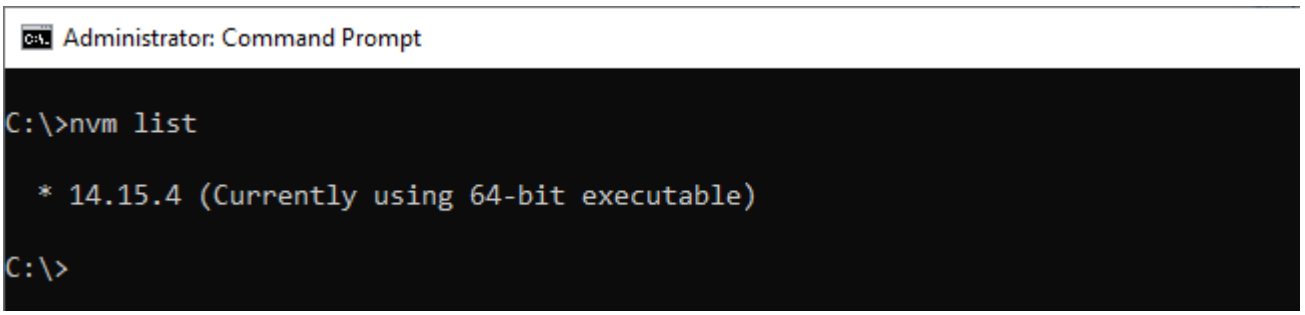
1. node -v
2. npm -v

```
C:\>nvm use 14.15.4
Now using node v14.15.4 (64-bit)

C:\>node -v
v14.15.4

C:\>npm -v
6.14.10

C:\>
```

Check the list of Node.js versions installed.

You can even check the total number of Node.js versions installed and running on your computer using the below command.

1. nvm list



Currently, we have the Node.js version (14.15.4) running on our device. We can also be able to install more different versions of Node.js on a computer if required (while building another Node.js project based on a different version). To install another version of Node.js, choose a version from the nvm list available and run the command given below.

1. nvm install 12.20.2

```
C:\>nvm install 12.20.2
Downloading node.js version 12.20.2 (64-bit)...
Complete
Creating C:\Users\JTP\AppData\Roaming\nvm\temp

Downloading npm version 6.14.11... Complete
Installing npm v6.14.11...

Installation complete. If you want to use this version, type

nvm use 12.20.2
```

Here, we have selected another Node version 12.20.2 to install, and it also downloads and installs compatible npm on our device.

Now, we have two different versions of Node.js and npm. To check the total number of installed Node.js version, run the following command:

1. nvm list

```
C:\>nvm list

  * 14.15.4 (Currently using 64-bit executable)
    12.20.2

C:\>
```

Now, we have two different versions of Node.js and npm on our computers.

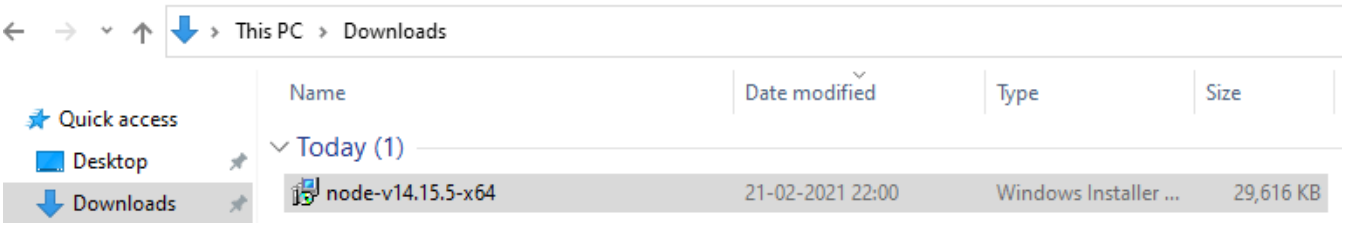
2. Install npm and Node.js using Node installer

Suppose you are not able to install, or you are facing some difficulty while installing npm using Node version manager on your device. The other way to install both npm and Node.js on your device is using **the Node.js installer.**

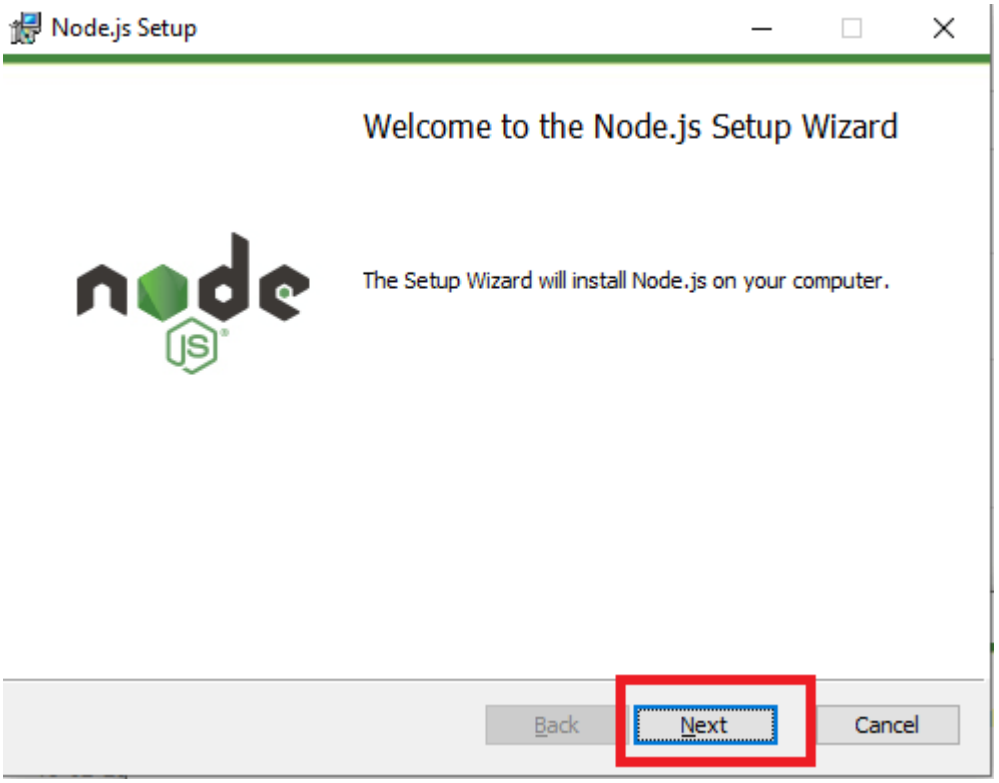
Navigate to the Node.js download section, where you will find Node.js installer for the different operating systems to install Node.js and npm together.

Depending on your computer operating system and its bits, download any supported Node.js installer.

1. We have Windows with the 64-bit operating system, and we are downloading a 64-bit Node.js installer for our device.
2. After downloading, run the setup and follow the installation instruction as shown on the screen.

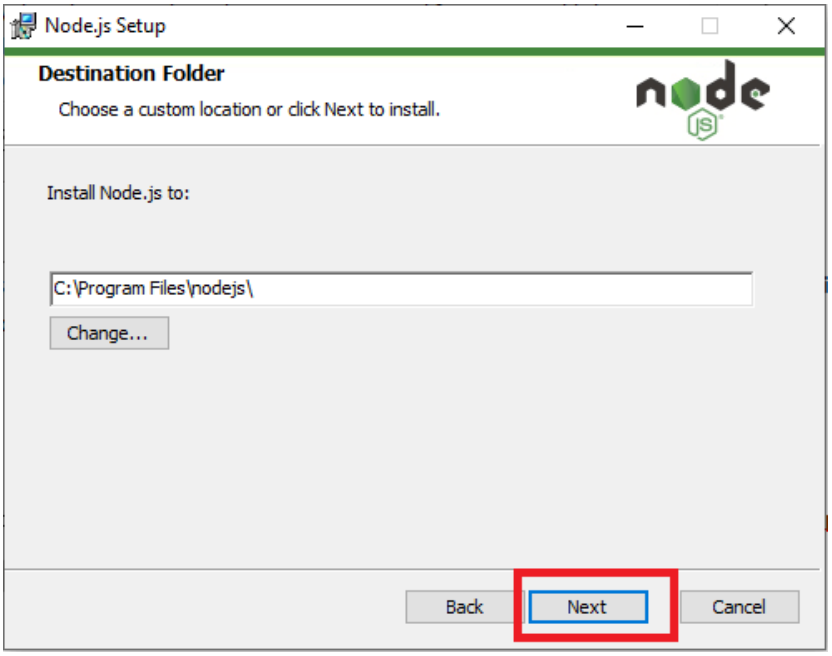


3. Click on the **Next** button to start the installation process.

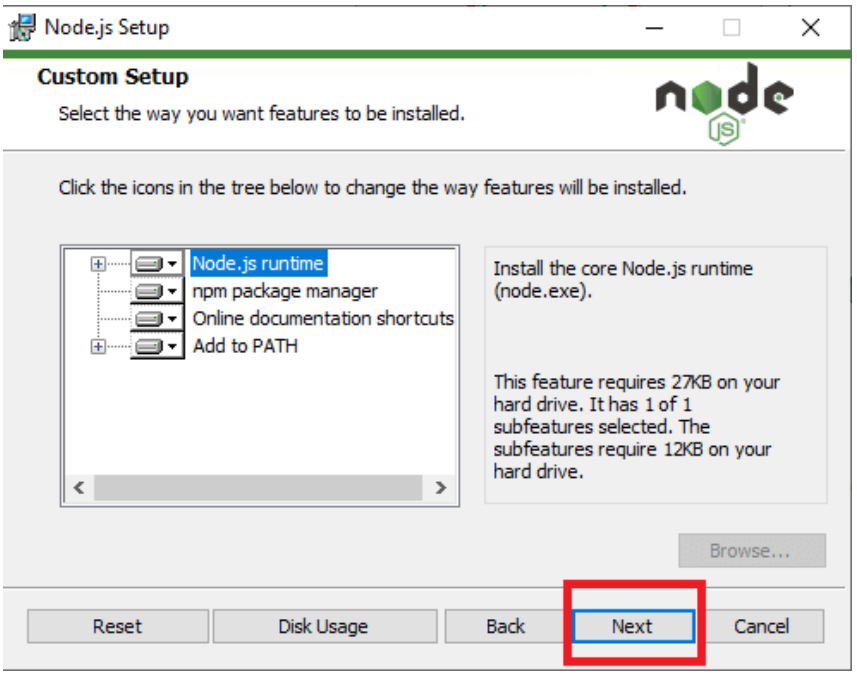


4. Read the end-user license agreement, accept it and click on the **Next** button.

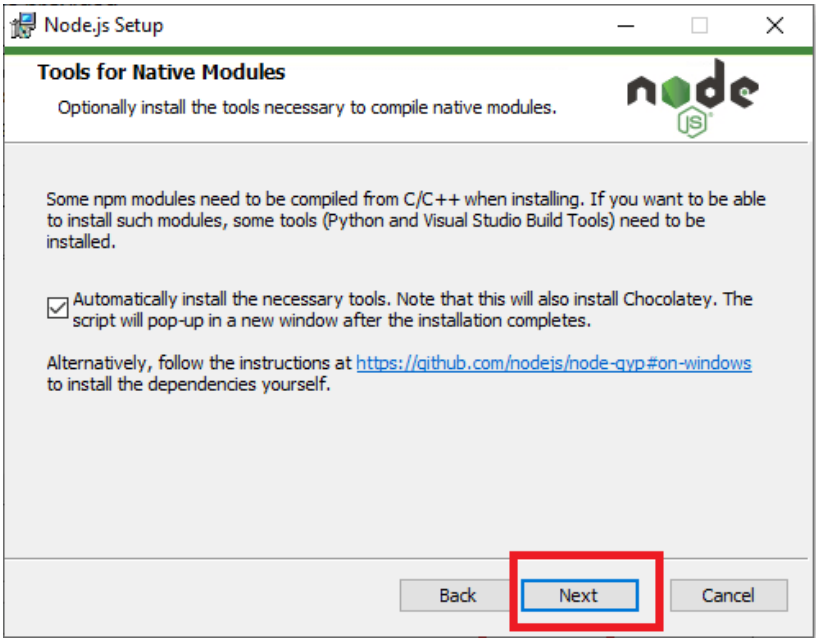
5. Choose a custom location of your Node.js and click on **Next** to install. We keep it as its default location.



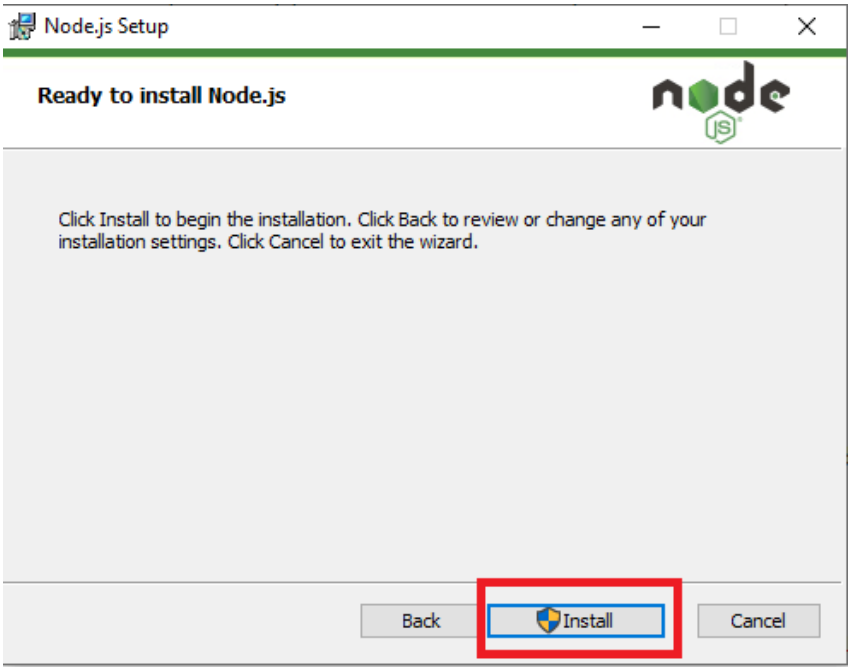
6. Select the feature of Node.js to get installed and click on **Next.** We keep it as its default feature provided.



7. If you want to install some other necessary tools (C/C++, Chocolaty) with your node.js and npm, checkmark the box and click on the **Next** button.

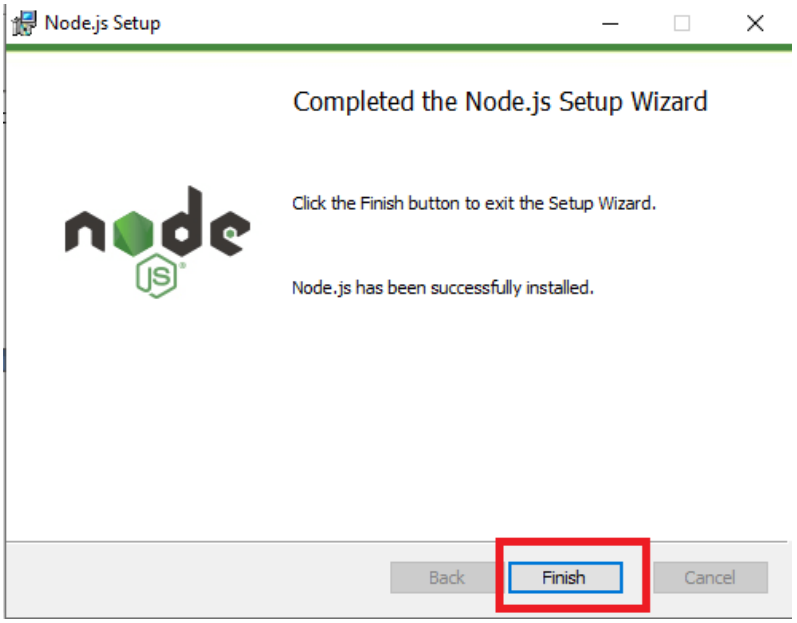


8. Now, click on the **Install** button to install Node.js, which also installs npm on your device.

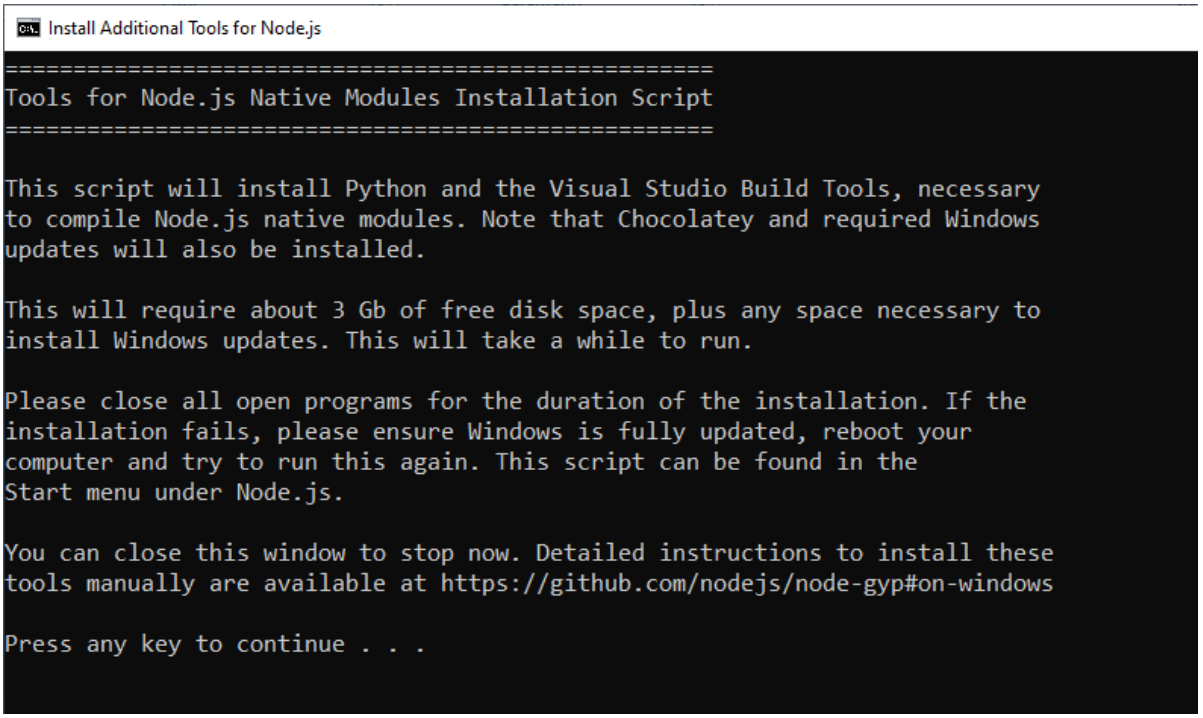


9. Click on **Yes** to install Node.js on the popup window.

10. Click on the **Finish** button to complete the installation.



The additional tools are ready to install, as we marked above to install additional tools automatically. Press any key to continue.



Run the following command on the command line to download the latest version of npm.

1. npm install -g npm

Properties of npm:

- All **npm** packages are defined in files called **package.json.**
- You must write the content of **package.json** in **JSON.**
- At least two fields must be present in the definition file: **name** and **version.**

package.json

package.json file will be generated when we run the **npm init** to initialize a JavaScript/Node.js project with the basic metadata information provided by developers:

name: it is the name of your JavaScript library/project

version: it is the version of your project

description: pass the project's description information

license: it is a project's license

When to use npm

- Use npm to adapt packages of code for your apps or incorporate packages as they are.
- npm can be used when you want to download standalone tools right away.
- Run packages without downloading using npx.
- Use npm when you need to share code with any npm user, anywhere.
- npm can be used when you want to restrict code to specific developers.
- Use npm to create organizations to coordinate package maintenance, coding, and developers.
- Use npm to manage multiple versions of code and code dependencies.
- Use npm to update applications easily when the underlying code is updated.
- Use npm to find other developers who are working on similar problems and projects.

SERVING FILES WITH THE HTTP MODULE

A basic necessity for most http servers is to be able to serve static files. Thankfully, it is not that hard to do in Node.js. First you read the file, then you serve the file. Here is an example of a script that will serve the files in the current directory:

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


http.createServer(function (req, res) {
  fs.readFile(__dirname + req.url, function (err,data) {
    if (err) {
      res.writeHead(404);
      res.end(JSON.stringify(err));
      return;
    }
    res.writeHead(200);
    res.end(data);
  });
```

}).listen(8080);

This example takes the path requested and it serves that path, relative to the local directory. This works fine as a quick solution; however, there are a few problems with this approach. First, this code does not correctly handle mime types. Additionally, a proper static file server should really be taking advantage of client side caching, and should send a "Not Modified" response if nothing has changed. Furthermore, there are security bugs that can enable a malicious user to break out of the current directory. (for example, GET /../../../).

Each of these can be addressed individually without much difficulty. You can send the proper mime type header. You can figure how to utilize the client caches. You can take advantage of path.normalize to make sure that requests don't break out of the current directory. But why write all that code when you can just use someone else's library?

There is a good static file server called node-static written by Alexis Sellier which you can leverage. Here is a script which functions similarly to the previous one:

```
var static = require('node-static');

var http = require('http');


var file = new(static.Server)(__dirname);


http.createServer(function (req, res) {

  file.serve(req, res);

}).listen(8080);
```

This is a fully functional file server that doesn't have any of the bugs previously mentioned. This is just the most basic set up, there are more things you can do if you look at the api. Also since it is an open source project, you can always modify it to your needs (and feel free to contribute back to the project.

INTRODUCTION TO EXPRESS FRAMEWORK

Node.js - Express Framework

Express Overview

Express is a minimal and flexible Node.js web application framework that provides a robust set of features to develop web and mobile applications. It facilitates the rapid development of Node based Web applications. Following are some of the core features of Express framework –

- Allows to set up middlewares to respond to HTTP Requests.
- Defines a routing table which is used to perform different actions based on HTTP Method and URL.
- Allows to dynamically render HTML Pages based on passing arguments to templates.

Installing Express

Firstly, install the Express framework globally using NPM so that it can be used to create a web application using node terminal.

$ npm install express --save

The above command saves the installation locally in the **node_modules** directory and creates a directory express inside node_modules. You should install the following important modules along with express −

- **body-parser** − This is a node.js middleware for handling JSON, Raw, Text and URL encoded form data.
- **cookie-parser** − Parse Cookie header and populate req.cookies with an object keyed by the cookie names.
- **multer** − This is a node.js middleware for handling multipart/form-data.

$ npm install body-parser --save
$ npm install cookie-parser --save
$ npm install multer --save

Hello world Example

Following is a very basic Express app which starts a server and listens on port 8081 for connection. This app responds with **Hello World!** for requests to the homepage. For every other path, it will respond with a **404 Not Found.**

```
var express = require('express');
var app = express();

app.get('/', function (req, res) {
  res.send('Hello World');
})

var server = app.listen(8081, function () {
  var host = server.address().address
  var port = server.address().port

  console.log("Example app listening at http://%s:%s", host, port)
})
```

Save the above code in a file named server.js and run it with the following command.

$ node server.js

You will see the following output −

Example app listening at http://0.0.0.0:8081

Open http://127.0.0.1:8081/ in any browser to see the following result.

Request & Response

Express application uses a callback function whose parameters are **request** and **response** objects.

```
app.get('/', function (req, res) {
// --
})
```

- Request Object − The request object represents the HTTP request and has properties for the request query string, parameters, body, HTTP headers, and so on.
- Response Object − The response object represents the HTTP response that an Express app sends when it gets an HTTP request.

You can print **req** and **res** objects which provide a lot of information related to HTTP request and response including cookies, sessions, URL, etc.

Basic Routing

We have seen a basic application which serves HTTP request for the homepage. Routing refers to determining how an application responds to a client request to a particular endpoint, which is a URI (or path) and a specific HTTP request method (GET, POST, and so on).

We will extend our Hello World program to handle more types of HTTP requests.

```
var express = require('express');
var app = express();

// This responds with "Hello World" on the homepage
app.get('/', function (req, res) {
  console.log("Got a GET request for the homepage");
  res.send('Hello GET');
})

// This responds a POST request for the homepage
app.post('/', function (req, res) {
```

```
  console.log("Got a POST request for the homepage");
  res.send('Hello POST');
})

// This responds a DELETE request for the /del_user page.
app.delete('/del_user', function (req, res) {
  console.log("Got a DELETE request for /del_user");
  res.send('Hello DELETE');
})

// This responds a GET request for the /list_user page.
app.get('/list_user', function (req, res) {
  console.log("Got a GET request for /list_user");
  res.send('Page Listing');
})

// This responds a GET request for abcd, abxcd, ab123cd, and so on
app.get('/ab*cd', function(req, res) {
  console.log("Got a GET request for /ab*cd");
  res.send('Page Pattern Match');
})

var server = app.listen(8081, function () {
  var host = server.address().address
  var port = server.address().port

  console.log("Example app listening at http://%s:%s", host, port)
})
```

Save the above code in a file named server.js and run it with the following command.

$ node server.js

You will see the following output −

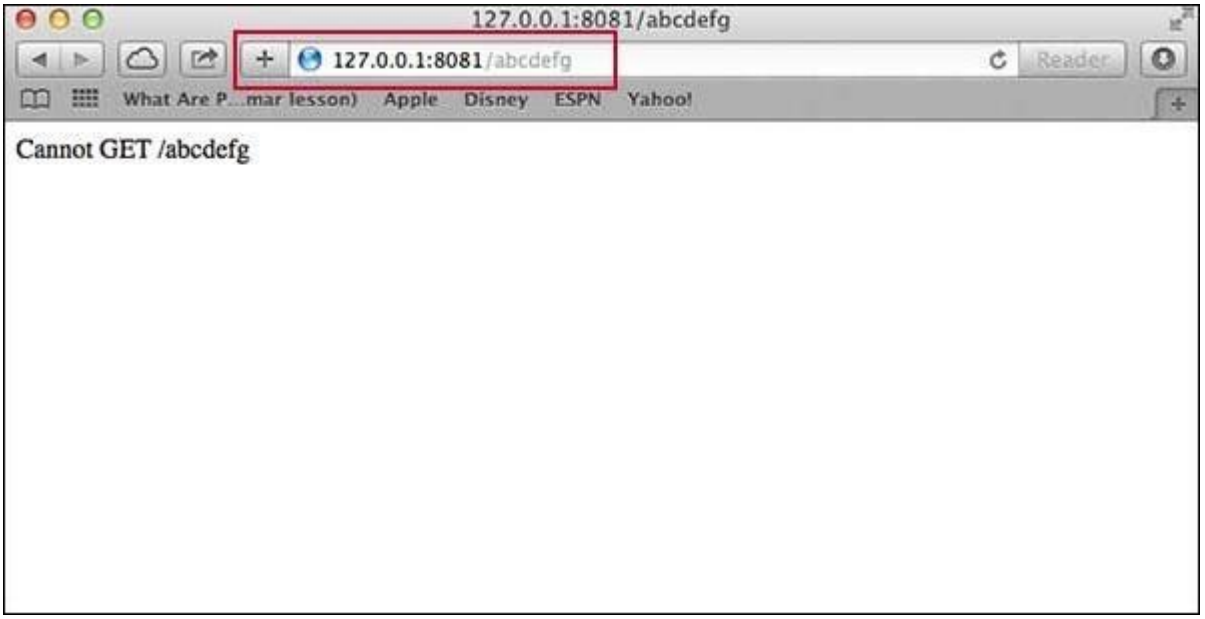
Example app listening at http://0.0.0.0:8081

Now you can try different requests at http://127.0.0.1:8081 to see the output generated by server.js. Following are a few screens shots showing different responses for different URLs.

Screen showing again http://127.0.0.1:8081/list_user

Screen showing again http://127.0.0.1:8081/abcd

Screen showing again http://127.0.0.1:8081/abcdefg



Serving Static Files

Express provides a built-in middleware **express.static** to serve static files, such as images, CSS, JavaScript, etc.

You simply need to pass the name of the directory where you keep your static assets, to the **express.static** middleware to start serving the files directly. For example, if you keep your images, CSS, and JavaScript files in a directory named public, you can do this −

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

We will keep a few images in **public/images** sub-directory as follows −

```
node_modules
server.js
public/
public/images
public/images/logo.png
```

Let's modify "Hello Word" app to add the functionality to handle static files.

```
var express = require('express');
var app = express();

app.use(express.static('public'));

app.get('/', function (req, res) {
   res.send('Hello World');
})

var server = app.listen(8081, function () {
   var host = server.address().address
   var port = server.address().port

   console.log("Example app listening at http://%s:%s", host, port)
})
```

Save the above code in a file named server.js and run it with the following command.

$ node server.js

Now open http://127.0.0.1:8081/images/logo.png in any browser and see observe following result.

GET Method

Here is a simple example which passes two values using HTML FORM GET method. We are going to use **process_get** router inside server.js to handle this input.

```
<html>
  <body>

    <form action = "http://127.0.0.1:8081/process_get" method = "GET">
      First Name: <input type = "text" name = "first_name">  <br>
      Last Name: <input type = "text" name = "last_name">
      <input type = "submit" value = "Submit">
    </form>

  </body>
</html>
```

Let's save above code in index.htm and modify server.js to handle home page requests as well as the input sent by the HTML form.

```
var express = require('express');
var app = express();

app.use(express.static('public'));
app.get('/index.htm', function (req, res) {
  res.sendFile( __dirname + "/" + "index.htm" );
})

app.get('/process_get', function (req, res) {
```

```
  // Prepare output in JSON format
  response = {
    first_name:req.query.first_name,
    last_name:req.query.last_name
  };
  console.log(response);
  res.end(JSON.stringify(response));
})

var server = app.listen(8081, function () {
  var host = server.address().address
  var port = server.address().port

  console.log("Example app listening at http://%s:%s", host, port)
})
```

Accessing the HTML document using http://127.0.0.1:8081/index.htm will generate the following form −

Top of Form

First Name:	
Last Name:	
SUBMIT	

Bottom of Form

Now you can enter the First and Last Name and then click submit button to see the result and it should return the following result −

```
{"first_name":"John","last_name":"Paul"}
```

POST Method

Here is a simple example which passes two values using HTML FORM POST method. We are going to use **process_get** router inside server.js to handle this input.

```
<html>
  <body>

    <form action = "http://127.0.0.1:8081/process_post" method = "POST">
      First Name: <input type = "text" name = "first_name"> <br>
      Last Name: <input type = "text" name = "last_name">
      <input type = "submit" value = "Submit">
    </form>

  </body>
</html>
```

Let's save the above code in index.htm and modify server.js to handle home page requests as well as the input sent by the HTML form.

```
var express = require('express');
var app = express();
var bodyParser = require('body-parser');

// Create application/x-www-form-urlencoded parser
var urlencodedParser = bodyParser.urlencoded({ extended: false })

app.use(express.static('public'));
app.get('/index.htm', function (req, res) {
  res.sendFile( __dirname + "/" + "index.htm" );
})

app.post('/process_post', urlencodedParser, function (req, res) {
  // Prepare output in JSON format
  response = {
    first_name:req.body.first_name,
    last_name:req.body.last_name
  };
  console.log(response);
  res.end(JSON.stringify(response));
})

var server = app.listen(8081, function () {
  var host = server.address().address
  var port = server.address().port

  console.log("Example app listening at http://%s:%s", host, port)
})
```

Accessing the HTML document using http://127.0.0.1:8081/index.htm will generate the following form −

Top of Form

First Name:	
Last Name:	
SUBMIT	

Bottom of Form

Now you can enter the First and Last Name and then click the submit button to see the following result −

```
{"first_name":"John","last_name":"Paul"}
```

File Upload

The following HTML code creates a file uploader form. This form has method attribute set to **POST** and enctype attribute is set to **multipart/form-data**

```
<html>
  <head>
    <title>File Uploading Form</title>
  </head>

  <body>
    <h3>File Upload:</h3>
    Select a file to upload: <br />

    <form action = "http://127.0.0.1:8081/file_upload" method = "POST"
      enctype = "multipart/form-data">
      <input type="file" name="file" size="50" />
      <br />
      <input type = "submit" value = "Upload File" />
    </form>

  </body>
</html>
```

Let's save above code in index.htm and modify server.js to handle home page requests as well as file upload.

```
var express = require('express');
var app = express();
var fs = require("fs");

var bodyParser = require('body-parser');
var multer  = require('multer');

app.use(express.static('public'));
app.use(bodyParser.urlencoded({ extended: false }));
app.use(multer({ dest: '/tmp/'}));

app.get('/index.htm', function (req, res) {
  res.sendFile( __dirname + "/" + "index.htm" );
})

app.post('/file_upload', function (req, res) {
  console.log(req.files.file.name);
  console.log(req.files.file.path);
  console.log(req.files.file.type);
  var file = __dirname + "/" + req.files.file.name;

  fs.readFile( req.files.file.path, function (err, data) {
    fs.writeFile(file, data, function (err) {
      if( err ) {
        console.log( err );
```

```
         } else {
           response = {
             message:'File uploaded successfully',
             filename:req.files.file.name
           };
         }

       console.log( response );
       res.end( JSON.stringify( response ) );
    });
  });
})

var server = app.listen(8081, function () {
  var host = server.address().address
  var port = server.address().port

  console.log("Example app listening at http://%s:%s", host, port)
})
```

Accessing the HTML document using http://127.0.0.1:8081/index.htm will generate the following form −

File Upload:
Select a file to upload:

Choose File No file chosen

Upload File

NOTE: This is just dummy form and would not work, but it must work at your server.

SERVER-SIDE RENDERING WITH TEMPLATING ENGINES

Templates - Server-Side Rendering

Regular Web Applications rely on templates to dynamically generate HTML pages on the server. These templates are text files that contain static content as well as a special syntax describing how the data should be inserted dynamically. During an HTTP request, the application loads and renders the template using the given contextual data and sends back the page to the client.

This technique is known as Server-Side Rendering (or SSR).

Here is an example of what a template might look like:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>{% title %}</title>
</head>
<body>
  {% for user in users %}
    * {{ user.name }}
  {% else %}
     No users have been found.
  {% endfor %}
</body>
</html>
```

Rendering Templates
FoalTS provides a minimalist template engine to render templates. This engine replaces all the occurrences of S myVariableName }} with the given values.

Here is an example showing how to use it:

templates/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>{{ title }}</title>
</head>
<body>
  Hello {{ name }}!
</body>
</html>
```

src/app/app.controller.ts

```
import { Get, render } from '@foal/core';

export class AppController {
  @Get('/')
  index() {
    return render('./templates/index.html', {
      name: 'Alix',
      title: 'Home'
    });
  }
}
```

Output (GET /)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Home</title>
</head>
<body>
  Hello Alix!
</body>
</html>
```

Using Another Template Engine

External template engines, such as EJS or pug, are also supported and can be configured for the current project using the configuration key settings.templateEngine.

Here is an example showing how to configure config/default.json (or config/default.yml) with twig, a JS implementation of the Twig PHP templating language.

```
npm install twig
```

```
module.exports = {
  settings: {
    templateEngine: "twig"
  }
}
```

Then the render function uses this engine under the hood to render the templates.

Note: Only Express compatible template engines are supported (which represents the large majority of those available on npm).

templates/index.html (Twig example)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Users</title>
</head>
<body>
  <ul>
    {% for user in users %}
      <li>{{ user.name }}</li>
    {% endfor %}
  </ul>
</body>
</html>
```

src/app/app.controller.ts (Twig example)

```
import { Get } from '@foal/core';

export class AppController {
  @Get('/')
  index() {
    return render('./templates/index.html', {
      users: [
        { name: 'John' },
        { name: 'Mary' }
      ]
    });
  }
}
```

Output (GET /) (Twig example)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Users</title>
</head>
<body>
  <ul>
    <li>John</li>
    <li>Mary</li>
  </ul>
</body>
</html>
```

ExpressJS - Serving static files

Static files are files that clients download as they are from the server. Create a new directory, **public**. Express, by default does not allow you to serve static files. You need to enable it using the following built-in middleware.

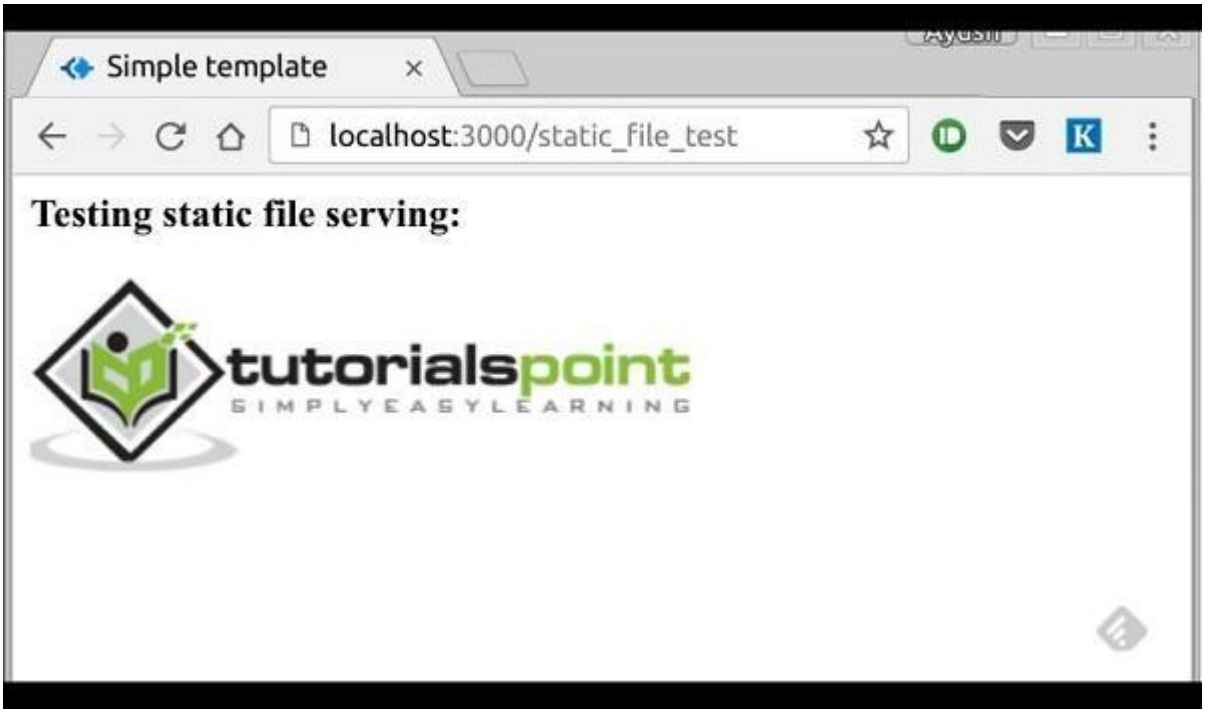
app.use(express.static('public'));

Note − Express looks up the files relative to the static directory, so the name of the static directory is not part of the URL.

Note that the root route is now set to your public dir, so all static files you load will be considering public as root. To test that this is working fine, add any image file in your new **public** dir and change its name to "**testimage.jpg**". In your views, create a new view and include this file like −

html
head
body
h3 Testing static file serving:
img(src = "/testimage.jpg", alt = "Testing Image

You should get the following output −



Multiple Static Directories

We can also set multiple static assets directories using the following program −

```
var express = require('express');
var app = express();

app.use(express.static('public'));
app.use(express.static('images'));

app.listen(3000);
```

Virtual Path Prefix

We can also provide a path prefix for serving static files. For example, if you want to provide a path prefix like **'/static'**, you need to include the following code in your **index.js** file −

```
var express = require('express');
var app = express();

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

app.listen(3000);
```

Now whenever you need to include a file, for example, a script file called main.js residing in your public directory, use the following script tag −

```
<script src = "/static/main.js" />
```

This technique can come in handy when providing multiple directories as static files. These prefixes can help distinguish between multiple directories.

Using Async Await in Node.js

Before Node version 7.6, the callbacks were the only official way provided by Node to run one function after another. As Node architecture is single-threaded and asynchronous, the community devised the callback functions, which would fire (or run) after the first function (to which the callbacks were assigned) run is completed.

Example of a Callback:

```
app.get('/', function(){
 function1(arg1, function(){
   ...
})
});
```

The problem with this kind of code is that this kind of situations can cause a lot of trouble and the code can get messy when there are several functions. This situation is called what is commonly known as a callback hell.
So, to find a way out, the idea of Promises and function chaining was introduced.

Example: Before async/await

```
function fun1(req, res){
  return request.get('http://localhost:3000')
   .catch((err) =>{
     console.log('found error');
}).then((res) =>{
   console.log('get request returned.');
});
```

Explanation:
The above code demos a function implemented with function chaining instead of callbacks. It can be observed that the code is now more easy to understand and readable. The code basically says that GET localhost:3000, catch the error if there is any; if there is no error then implement the following statement:
console.log(‘get request returned.’);

With Node v8, the async/await feature was officially rolled out by the Node to deal with Promises and function chaining. The functions need not to be chained one after another, simply await the function that returns the Promise. But the function async needs to be declared before awaiting a function returning a Promise. The code now looks like below.

Example: After async/await

```
async function fun1(req, res){
  let response = await request.get('http://localhost:3000');
```

```
    if (response.err) { console.log('error');}
    else { console.log('fetched response');
}
```

Explanation:
The code above basically asks the javascript engine running the code to wait for the request.get() function to complete before moving on to the next line to execute it.
The request.get() function returns a Promise for which user will await . Before async/await, if it needs to be made sure that the functions are running in the desired sequence, that is one after the another, chain them one after the another or register callbacks.
Code writing and understanding becomes easy with async/await as can be observed from both the examples.

FETCHING JSON FROM EXPRESS

Write the ex: program 4 th and explain it.

www.BrainKart.com

Anna University

for Affilated Engineering College - 2021 Regulation

Information Technology

1st Semester | 2nd Semester | 3rd Semester | 4th Semester

5th Semester | 6th Semester | 7th Semester | 8th Semester

Click on Subject/Paper under Semester to enter.

1st Semester

- Professional English - I - HS3152
- Matrices and Calculus - MA3151
- Engineering Physics - PH3151
- Engineering Chemistry - CY3151
- Problem Solving and Python Programming - GE3151

2nd Semester

- Professional English - II - HS3252
- Statistics and Numerical Methods - MA3251
- Engineering Graphics - GE3251
- Physics for Information Science - PH3256
- Basic Electrical and Electronics Engineering - BE3251
- Programming in C - CS3251

3rd Semester

- Discrete Mathematics - MA3354
- Digital Principles and Computer Organization - CS3351
- Foundation of Data Science - CS3352
- Data Structures and Algorithms - CD3291
- Object Oriented Programming - CS3391

4th Semester

- Environmental Sciences and Sustainability - GE3451
- Theory of Computation - CS3452
- Artificial Intelligence and Machine Learning - CS3491
- Database Management System - CS3492
- Web Essentials - IT3401
- Introduction to Operating Systems - CS3451

5th Semester

- Computer Networks - CS3591
- Full Stack Web Development - IT3501
- Distributed Computing - CS3551
- Embedded Systems and IoT - CS3691
- Elective 1
- Elective 2

6th Semester

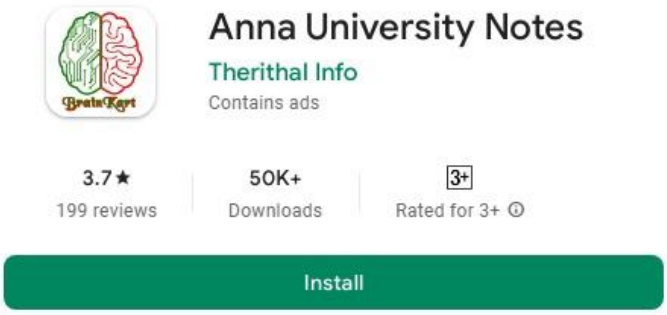
- Object Oriented Software Engineering - CCS356
- Open Elective-1
- Elective-3
- Elective-4
- Elective-5
- Elective-6

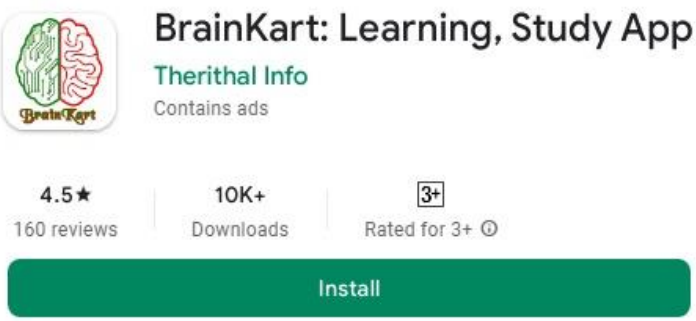
7th Semester

- Human Values and Ethics - GE3791
- Open Elective 2
- Open Elective 3
- Open Elective 4
- Management Elective

8th Semester

- Project Work / Intership





All Computer Engg Subjects - [B.E., M.E.,]		(Click on Subjects to enter)
Programming in C	Computer Networks	Operating Systems
Programming and Data Structures I	Programming and Data Structure II	Problem Solving and Python Programming
Database Management Systems	Computer Architecture	Analog and Digital Communication
Design and Analysis of Algorithms	Microprocessors and Microcontrollers	Object Oriented Analysis and Design
Software Engineering	Discrete Mathematics	Internet Programming
Theory of Computation	Computer Graphics	Distributed Systems
Mobile Computing	Compiler Design	Digital Signal Processing
Artificial Intelligence	Software Testing	Grid and Cloud Computing
Data Ware Housing and Data Mining	Cryptography and Network Security	Resource Management Techniques
Service Oriented Architecture	Embedded and Real Time Systems	Multi - Core Architectures and Programming
Probability and Queueing Theory	Physics for Information Science	Transforms and Partial Differential Equations
Technical English	Engineering Physics	Engineering Chemistry
Engineering Graphics	Total Quality Management	Professional Ethics in Engineering
Basic Electrical and Electronics and Measurement Engineering	Problem Solving and Python Programming	Environmental Science and Engineering