

# NodeJS

ashok.dudhat@devugees.org

# How To Install the Distro-Stable Version for Ubuntu

Ubuntu 16.04 contains a version of Node.js in its default repositories that can be used to easily provide a consistent experience across multiple systems. At the time of writing, the version in the repositories is v4.2.6. This will not be the latest version, but it should be quite stable and sufficient for quick experimentation with the language. In order to get this version, we just have to use the `apt` package manager. We should refresh our local package index first, and then install from the repositories:

```
$ sudo apt-get update
$ sudo apt-get install nodejs
```

If the package in the repositories suits your needs, this is all you need to do to get set up with Node.js. In most cases, you'll also want to also install `npm`, which is the Node.js package manager. You can do this by typing:

```
$ sudo apt-get install npm
```

This will allow you to easily install modules and packages to use with Node.js. Because of a conflict with another package, the executable from the Ubuntu repositories is called `nodejs` instead of `node`. Keep this in mind as you are running software. To check which version of Node.js you have installed after these initial steps, type:

```
$ nodejs -v
```

# How To Install Using a PPA

An alternative that can get you a more recent version of Node.js is to add a PPA (personal package archive) maintained by NodeSource. This will have more up-to-date versions of Node.js than the official Ubuntu repositories, and allows you to choose between Node.js v4.x (the older long-term support version, which will be supported until April of 2018), Node.js v6.x (supported until April of 2019), and Node.js v8.x (the current LTS version, supported until December of 2019).

First, you need to install the PPA in order to get access to its contents. Make sure you're in your home directory, and use `curl` to retrieve the installation script for your preferred version, making sure to replace `8.x` with your preferred version string (if different):

# Debian and Ubuntu based Linux distributions

Also including: **Linux Mint**, **Linux Mint Debian Edition (LMDE)**, **elementaryOS**, **bash on Windows** and others.

Node.js is available from the [NodeSource](#) Debian and Ubuntu binary distributions repository (formerly [Chris Lea's](#) Launchpad PPA). Support for this repository, along with its scripts, can be found on GitHub at [nodesource/distributions](#).

**NOTE:** If you are using Ubuntu Precise or Debian Wheezy, you might want to read about [running Node.js >= 6.x on older distros](#).

```
curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -  
sudo apt-get install -y nodejs
```

Alternatively, for Node.js 10:

```
curl -sL https://deb.nodesource.com/setup_10.x | sudo -E bash -  
sudo apt-get install -y nodejs
```

*Optional:* install build tools

To compile and install native addons from npm you may also need to install build tools:

```
sudo apt-get install -y build-essential
```

To check which version of Node.js you have installed after these initial steps, type:

```
$ nodejs -v
```

Output

```
v8.10.0
```

The `nodejs` package contains the `nodejs` binary as well as `npm`, so you don't need to install `npm` separately.

`npm` uses a configuration file in your home directory to keep track of updates. It will be created the first time you run `npm`. Execute this command to verify that `npm` is installed and to create the configuration file:

```
$ npm -v
```

Output

```
5.6.0
```

## Check the npm Help Documentation

A good way to start using npm is to read the npm help page or the npm documentation. To check the npm help page, enter the following command:

```
npm help
```

```
# npm help
```

Usage: npm <command>

where <command> is one of:

access, add-user, adduser, apihelp, author, bin, bugs, c,  
cache, completion, config, ddp, dedupe, deprecate, dist-tag,  
dist-tags, docs, edit, explore, faq, find, find-dupes, get,  
help, help-search, home, i, info, init, install, issues, la,  
link, list, ll, ln, login, logout, ls, outdated, owner,  
pack, ping, prefix, prune, publish, r, rb, rebuild, remove,  
repo, restart, rm, root, run-script, s, se, search, set,  
show, shrinkwrap, star, stars, start, stop, t, tag, team,  
test, tst, un, uninstall, unlink, unpublish, unstar, up,  
update, upgrade, v, version, view, whoami

```
npm -h      quick help on
npm -l      display full usage info
npm faq     commonly asked questions
npm help    search for help on
npm help npm involved overview
```

Specify configs in the ini-formatted file:

/root/.npmrc

or on the command line via: npm <command> --key value

Config info can be viewed via: npm help config

```
npm@4.2.0 /usr/lib/node_modules/npm <command>
```

# Node.js Web Server

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 and Apache is a web server for PHP or Java web applications.

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



# Create Node.js Web Server

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

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

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

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

    //handle incoming requests here..

});

server.listen(5000); //3 - listen for any incoming requests

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

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

```
$ node server.js  
Node.js web server at port 5000 is running..
```

# 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.

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

var server = http.createServer(function (req, res) { //create web server
  if (req.url == '/') { //check the URL of the current request

    // set response header
    res.writeHead(200, { 'Content-Type': 'text/html' });

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

  }
  else if (req.url == "/student") {

    res.writeHead(200, { 'Content-Type': 'text/html' });
    res.write('<html><body><p>This is student Page.</p></body></html>');
    res.end();

  }
  else if (req.url == "/admin") {

    res.writeHead(200, { 'Content-Type': 'text/html' });
    res.write('<html><body><p>This is admin Page.</p></body></html>');
    res.end();

  }
  else
    res.end('Invalid Request!');

});

server.listen(5000); //6 - listen for any incoming requests

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

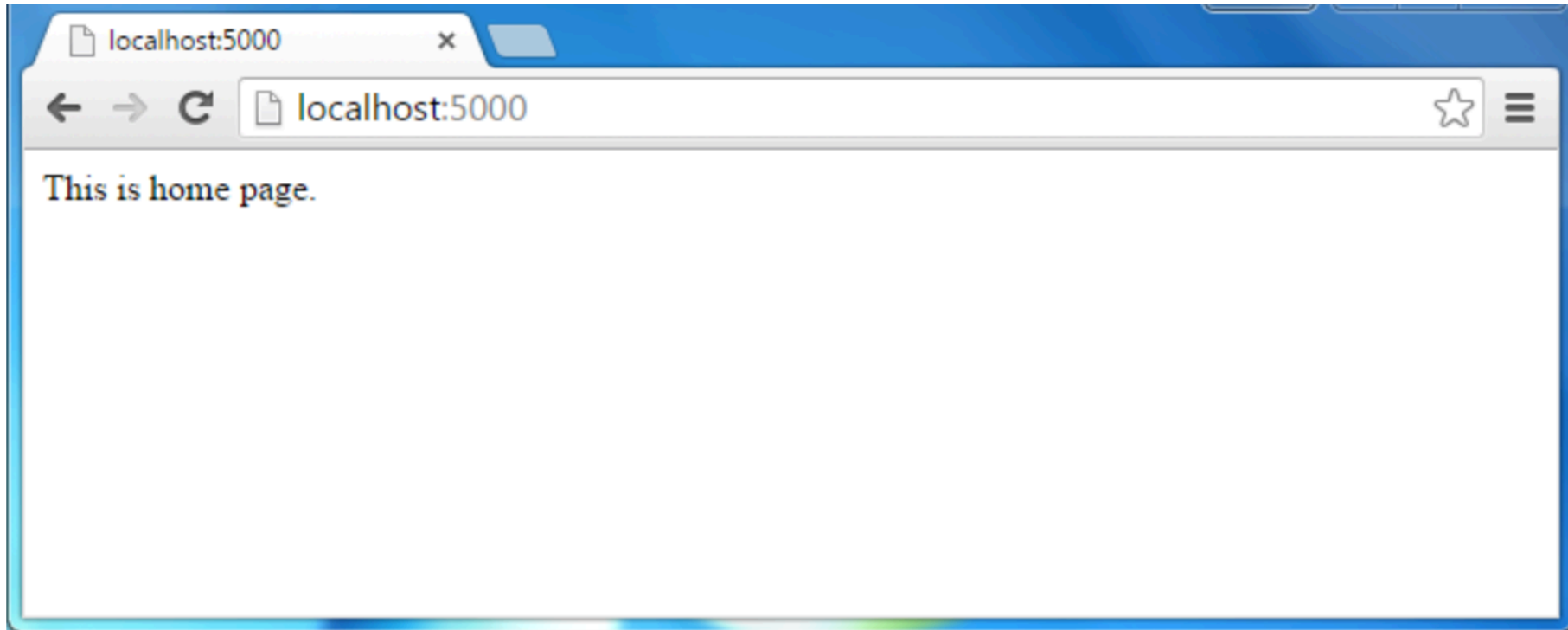
In the above example, `req.url` is used to check the url of the current request and based on that it sends the response. To send a response, first it sets the response header using `writeHead()` method and then writes a string as a response body using `write()` method. Finally, Node.js web server sends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

Now, run the above web server as shown below.

```
node server.js
```

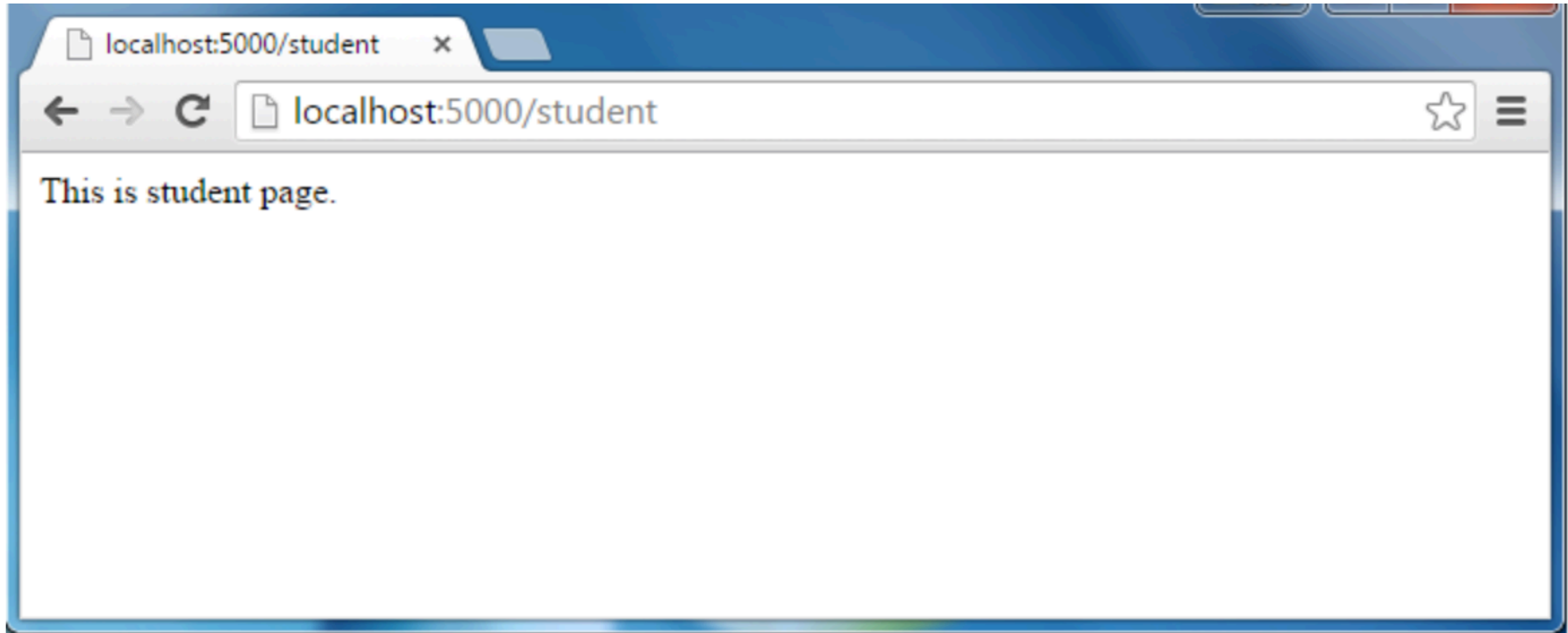
```
Node.js web server at port 5000 is running..
```

Point your browser to *http://localhost:5000* and see the following result.



Node.js Web Server Response

The same way, point your browser to *http://localhost:5000/student* and see the following result.



Node.js Web Server Response

# Sending JSON Response

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

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

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

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

server.listen(5000);

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



# Frameworks for Node.js

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

The following table lists frameworks for Node.js.

Open-Source Framework	Description
<a href="#">Express.js</a>	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.
<a href="#">Geddy</a>	Geddy is a simple, structured web application framework for Node.js based on MVC architecture.
<a href="#">Locomotive</a>	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.
<a href="#">Koa</a>	Koa is a new web framework designed by the team behind Express, which aims to be a smaller, more expressive, and more robust foundation for web applications and APIs.
<a href="#">Total.js</a>	Totaljs is free web application framework for building web sites and web applications using JavaScript, HTML and CSS on Node.js
<a href="#">Hapi.js</a>	Hapi is a rich Node.js framework for building applications and services.
<a href="#">Keystone</a>	Keystone is the open source framework for developing database-driven websites, applications and APIs in Node.js. Built on Express and MongoDB.

<a href="#">Derbyjs</a>	Derby support single-page apps that have a full MVC structure, including a model provided byÂ Racer, a template and styles based view, and controller code with application logic and routes.
<a href="#">Sails.js</a>	Sails makes it easy to build custom, enterprise-grade Node.js apps. It is designed to emulate the familiar MVC pattern of frameworks like Ruby on Rails, but with support for the requirements of modern apps: data-driven APIs with a scalable, service-oriented architecture. It's especially good for building chat, realtime dashboards, or multiplayer games; but you can use it for any web application project - top to bottom.
<a href="#">Meteor</a>	Meteor is a complete open source platform for building web and mobile apps in pure JavaScript.
<a href="#">Mojito</a>	This HTML5 framework for the browser and server from Yahoo offers direct MVC access to the server database through the local routines. One clever feature allows the code to migrate. If the client can't run JavaScript for some reason, Mojito will run it on the server -- a convenient way to handle very thin clients.
<a href="#">Restify</a>	Restify is a node.js module built specifically to enable you to build correct REST web services.
<a href="#">Loopback</a>	Loopback is an open-source Node.js API framework.
<a href="#">ActionHero</a>	actionhero.js is a multi-transport Node.JS API Server with integrated cluster capabilities and delayed tasks.
<a href="#">Frisby</a>	Frisby is a REST API testing framework built on node.js and Jasmine that makes testing API endpoints easy, fast, and fun.
<a href="#">Chocolate.js</a>	Chocolate is a simple webapp framework built on Node.js using Coffeescript.

# Express.js

"Express is a fast, unopinionated minimalist web framework for Node.js" - official web site: [Expressjs.com](https://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

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

# 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.

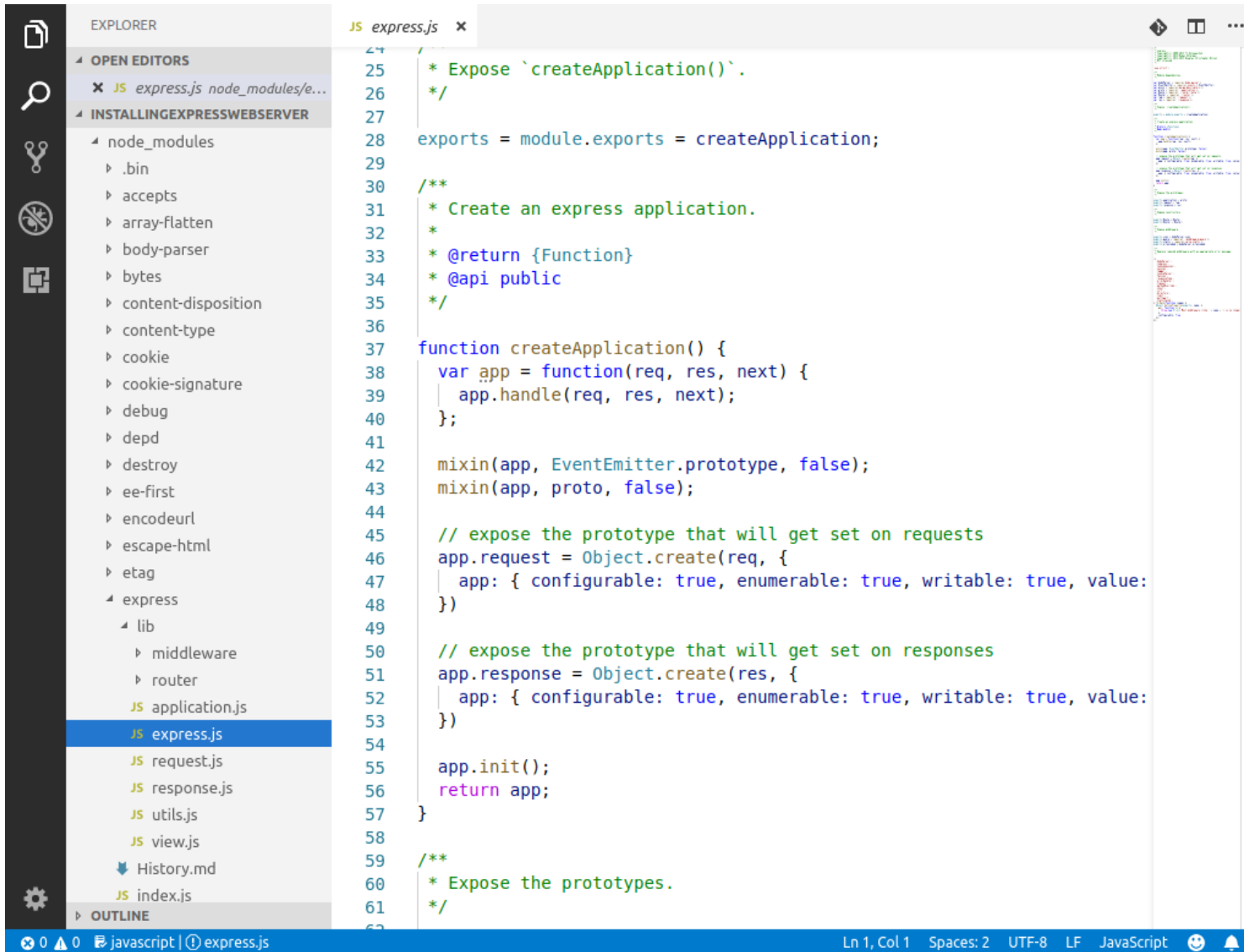
```
npm install -g express
```

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

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

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

# What include Express.js file?



The screenshot shows the Visual Studio Code editor with the Express.js source code open in the `express.js` file. The Explorer sidebar on the left shows the file structure of the `node_modules` directory, with `express.js` selected under the `express` folder. The main editor area displays the code for `express.js`, which includes comments and function definitions for creating and exposing an Express application.

```
25  * Expose `createApplication()`.
26  */
27
28  exports = module.exports = createApplication;
29
30  /**
31   * Create an express application.
32   *
33   * @return {Function}
34   * @api public
35   */
36
37  function createApplication() {
38    var app = function(req, res, next) {
39      app.handle(req, res, next);
40    };
41
42    mixin(app, EventEmitter.prototype, false);
43    mixin(app, proto, false);
44
45    // expose the prototype that will get set on requests
46    app.request = Object.create(req, {
47      app: { configurable: true, enumerable: true, writable: true, value:
48    });
49
50    // expose the prototype that will get set on responses
51    app.response = Object.create(res, {
52      app: { configurable: true, enumerable: true, writable: true, value:
53    });
54
55    app.init();
56    return app;
57  }
58
59  /**
60   * Expose the prototypes.
61   */
```

The status bar at the bottom indicates the current file is `express.js` in the `javascript` language, with 1 line and 1 column selected, using 2 spaces, UTF-8 encoding, and LF line endings.

# Environment Variables

- Global variables specific to the environment (server) our code is living in.
- Different servers can have different variable settings, and we can access those values in code.

example: `process.env.PORT`



# Create app.js using express

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

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

```
var port = process.env.PORT || 3000;
```

```
app.listen(port);
```

# HTTP Methods

- Specifies the type of action the request wishes to make.
- GET, POST, DELETE and others. Also called verbs.

example.

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

```
app.get('/api', function(req, res) {  
    res.json({ firstname: 'John', lastname: 'Doe' });  
});
```

# Routing

**Routing** refers to how an application's endpoints (URIs) respond to client requests.

You define routing using methods of the Express `app` object that correspond to HTTP methods; for example, `app.get ( )` to handle GET requests and `app.post` to handle POST requests.

```
app.get('/person/:id', function(req, res) {  
    res.send('<html><head></head><body><h1>Person: ' + req.params.id + '</  
h1></body></html>');  
});
```

# Route Methods

A route method is derived from one of the HTTP methods, and is attached to an instance of the `express` class.

The following code is an example of routes that are defined for the GET and the POST methods to the root of the app.

```
// GET method route
app.get('/', function (req, res) {
  res.send('GET request to the homepage')
})

// POST method route
app.post('/', function (req, res) {
  res.send('POST request to the homepage')
})
```

# Static Files and Middleware

Static : Not Dynamic.

In other words, not processed by code in any way. For example HTML, CSS and image files are 'static' files.

Middleware : Code that sits between two layers of software.

In the case of Express, sitting between the request and the response.

# 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.

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

var port = process.env.PORT || 3000;

//setting middleware
app.use(express.static(__dirname + 'public')); //Serves resources from public folder

var server = app.listen(5000);
```

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.

# Middleware

Express is a routing and middleware web framework that has minimal functionality of its own: An Express application is essentially a series of middleware function calls.

**Middleware** functions are functions that have access to the request object (`req`), the response object (`res`), and the next middleware function in the application's request-response cycle. The next middleware function is commonly denoted by a variable named **`next`**.

Middleware functions can perform the following tasks:

- Execute any code.
- Make changes to the request and the response objects.
- End the request-response cycle.
- Call the next middleware function in the stack.

If the current middleware function does not end the request-response cycle, it must call `next ( )` to pass control to the next middleware function. Otherwise, the request will be left hanging.

An Express application can use the following types of middleware:

- Application-level middleware
- Router-level middleware
- Error-handling middleware
- Built-in middleware
- Third-party middleware

You can load application-level and router-level middleware with an optional mount path. You can also load a series of middleware functions together, which creates a sub-stack of the middleware system at a mount point.

# Application-level middleware

Bind application-level middleware to an instance of the app object by using the `app.use()` and `app.METHOD()` functions, where `METHOD` is the HTTP method of the request that the middleware function handles (such as GET, PUT, or POST) in lowercase.

This example shows a middleware function with no mount path. The function is executed every time the app receives a request.

```
var app = express()

app.use(function (req, res, next) {
  console.log('Time:', Date.now())
  next()
})
```



# Task

Example of a server that reads/creates and deletes files.

- Create a folder 'files' and put halloworld.txt, hallomars.txt, hallomoon.txt files inside this folder.
- Use the following 'express route methods' and 'fs' lib for this task.
- app.get route inside use → readFile method of fs
- app.delete route inside use → unlink method of fs
- app.post route inside use → writeFile method of fs

# Multer

[Multer](#) is a node.js middleware for handling multipart/form-data, which is primarily used for uploading files. It is written on top of busboy for maximum efficiency.

**NOTE:** Multer will not process any form which is not multipart (multipart/form-data).

# Installation

```
$ npm install --save multer
```

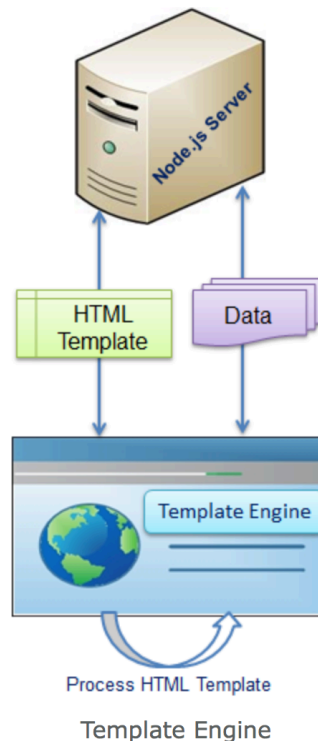
```
$ npm i multer
```

Let check example of the upload file.

# Templates and Template Engines

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.



As per the 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.

- [Jade](#)
- [Vash](#)
- [EJS](#)
- [Mustache](#)
- [Dust.js](#)
- [Nunjucks](#)
- [Handlebars](#)
- [atpl](#)
- [haml](#)

# Advantages of Template engine in Node.js

- 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 JavaScript Templating)

- Install EJS : `npm install ejs - - save`

# Querystring and Post Parameters

Browser send query string:

Example:

GET /?id=4&page=3 HTTP/1.1

Host:www.xyz.com Cookie:

username =abc; name=Tony



# Querystring and Post Parameters

Browser POST query string request:

Example: we have form with username and password with submit button.

POST / HTTP /1.1

Host:www.xyz.com

Content-Type: application/x-www-form-urlencoded

Cookie: num=4;page=2

username=Tony&password=pwd

# Querystring and Post Parameters

Browser Side JSON format

POST / HTTP /1.1

Host:www.xyz.com

Content-Type: application/json

Cookie: num=4;page=2

```
{  
  "username" : "Tony",  
  "password" : "pwd"  
}
```

# Node.js body parsing middleware.

Parse incoming request bodies in a middleware before your handlers, available under the `req.body` property.

This module provides the following parsers:

- JSON body parser
- Raw body parser
- Text body parser
- URL-encoded form body parser

## Installation

```
$ npm install body-parser
```

## API

```
var bodyParser = require('body-parser')
```

# RESTful APIs and JSON

REST : An Architectural Style For Building APIs.

Stands for 'Representational State Transfer'.  
We decide that HTTP verbs and URLs mean something.

# Part 1 : Task Creating RESTful

**users.json:**

```
[
  {
    "name": "mahesh",
    "password": "password1",
    "profession": "teacher",
    "id": 1
  },
  {
    "name" : "suresh",
    "password" : "password2",
    "profession" : "librarian",
    "id": 2
  },
  {
    "name" : "ramesh",
    "password" : "password3",
    "profession" : "clerk",
    "id": 3
  }
]
```

# Part 1 : Task Creating RESTful

**Based users.json information, please create following RESTful APIs.**

S. N.	URI	HTTP Method	POST body	Result
1	listUsers	GET	empty	Show list of all the users.
2	addUser	POST	JSON String	Add details of new user.
3	deleteUser	DELETE	JSON String	Delete an existing user.
4	:id	GET	empty	Show details of a user.

## Part 2: Use EJS template concept with created API

- Create template for showing list of users
- Create template for adding user

# views/users.ejs

ID	UserName	Password	Profession	Action
1	Mahesh	password1	teacher	Delete
2	Suresh	password2	librarian	Delete
3	Ramesh	password3	Clerk	Delete



# views/creatuser.ejs

Enter unique ID

Enter Username

Enter Password

Enter Profession

Submit

# Structuring an App

You can create a new Express application using the Express Generator tool. The Express Generator is shipped as an NPM module and installed by using the NPM command line tool npm.

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

Display the command options with the `-h` option:

```
$ express -h
```

For example, the following creates an Express app named myapp. The app will be created in a folder named myapp in the current working directory and the view engine will be set to Pug:

```
$ express --view=pug myapp
```

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

Then install dependencies:

```
$ cd myapp
$ npm install
```

On MacOS or Linux, run the app with this command:

```
$ DEBUG=myapp:* npm start
```