# Express Framework in Node.js

Express is a small framework and utility for building web applications, providing a huge amount of powerful features for developing web and mobile applications. It is easy to develop fast applications based on Node.js for Web applications. Below are the basic features of Express framework.

## Introducing Express Framework

Express is a small framework and utility for building web applications, providing a huge amount of powerful features for developing web and mobile applications. It is easy to develop fast applications based on Node.js for Web applications. Below are the basic features of Express framework.

Allow to set up intermediate classes to return HTTP requests.

The routing table definition can be used with different actions based on HTTP method and URL.

Allow returning HTML pages based on input parameters to the template.

## Install Express Framework

First, installing Express framework using npm is as follows:

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

The above command saves the installation in the node_modules directory and creates the express folder within that directory. Below are the important module components installed with express:

**body-parser** - This is an intermediate node node.js for handling JSON, raw data, text and URL encoding.

**cookie-parser** - Convert Cookie header and distribution to req.cookies

**multer** - This is an intermediate component in node.js to handle the multipart / form-data section.

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

Helloworld application example in Node.js

Here is a very basic example of Express that illustrates how to turn on the Server and listen to connections on port 3000. This application returns Hello World! for requests to the homepage. For other paths, it will return a 404 Not Found.

Create server.js with the following content:

```
var express = require ( 'express' ); var app = express (); app . get ( '/
```

Run server.js to see the result.

```
$ node server.js
```

You will see the results appear:

```
Cavalier Node.js is listening to the ear: http://0.0.0.0:8081
```

Open http://127.0.0.1:8081/ in any browser and see the results.


# Request & Response object in Node.js

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

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

You can refer to the details of these two objects below:

1. Request object - This object represents an HTTP request and has properties for requests such as query strings, parameters, body, HTTP headers and others.
2. Response object - This object represents the HTTP response sent by the Express application when it receives an HTTP request.

You can print **req** and **res** objects to provide a large amount of information related to HTTP requests and return cookies, sessions, URLs .

# Basic routing

Above, you have just watched a basic application that the HTTP Server requests to a home page. Routing involves identifying an application that returns a Client Request to a specific Endpoint, which is a URI path and returns an HTTP request (GET, POST and other methods).

Based on the above Hello World program, I will develop some additional functions to handle HTTP requests.

```
var express = require ( 'express' ); var app = express (); // Phuong thuc
```

Save the code in server.js and run this file with the following command:

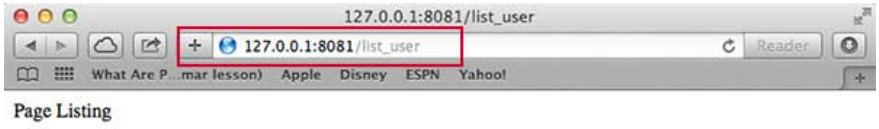
```
$ node server.js
```

Check the result:

```
Cavalier Node.js is listening to the ear: http://0.0.0.0:8081
```

Now, you can try other Requests at http://127.0.0.1:8081 to see the results created by server.js. Here are some screens showing different responses with different URLs.

Screen results for http://127.0.0.1:8081/list_user




Screen results for http://127.0.0.1:8081/abcd


Screen results for http://127.0.0.1:8081/abcdefg


# For static files

Express provides **express.static** middleware utilities to serve static files like images, CSS, Javascript, .

Basically, you just need to pass the directory name where you keep these files, **express.static** will use that file directly. For example, if you want to keep images, CSS and Javascript in the public directory, you can do the following:

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

Suppose I keep some images in the subdirectory **public / images** as follows:

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

Modify the "Hello Word" application to add some additional features to handle static files:

```
var  express  = require ( 'express' ); var  app  =  express ();  app . use ( exp
```

Save the code in server.js and run this file with the following command:

```
$ node server.js
```

Now open the browser and type the address http://127.0.0.1:8081/images/logo.png to see the result.

# Example GET method

Here is a simple example to pass two values ??using the HTML FORM with the GET method. I will use **process_get** in server.js to handle the input.

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

Save the above code in index.htm and modify server.js as follows.

```
var  express  = require ( 'express' ); var  app  =  express ();  app . use ( exp
```

Open the browser and type the address http://127.0.0.1:8081/index.htm to see the result:

```
{ "first_name" : "Hoang" , "last_name" : "Nguyen Manh" }
```

# Example POST method

Below is a basic example to pass two values ??using HTML form using POST method. I will use **process_post** in server.js to handle this input.

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

Save the above code in index.htm and modify server.js as follows:

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

Open the browser and type the address http://127.0.0.1:8081/index.htm to see the result:

```
{ "first_name" : "Hoang" , "last_name" : "Nguyen Manh" }
```

# For example File Upload

Here is HTML code to create a File Upload Form. This form has properties set to **POST** method and encryption attribute to set **multipart / form-data** .

```
File Uploading Form



File Upload:  Select a file to upload:
span="">  />
```

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

Save the above code in index.htm and modify server.js as follows:

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

Now open the browser and type the address http://127.0.0.1:8081/index.htm to see the result:

# Example of Cookie management

You can send Cookies to Node.js Server. The following example illustrates how to print all Cookies sent by the Client.

```
var express = require ( 'express' ) var cookieParser = require ( 'cookie-pa
```

**According to Tutorialspoint**

Previous article: Web Module in Node.js

Next article: RESTful API in Node.js