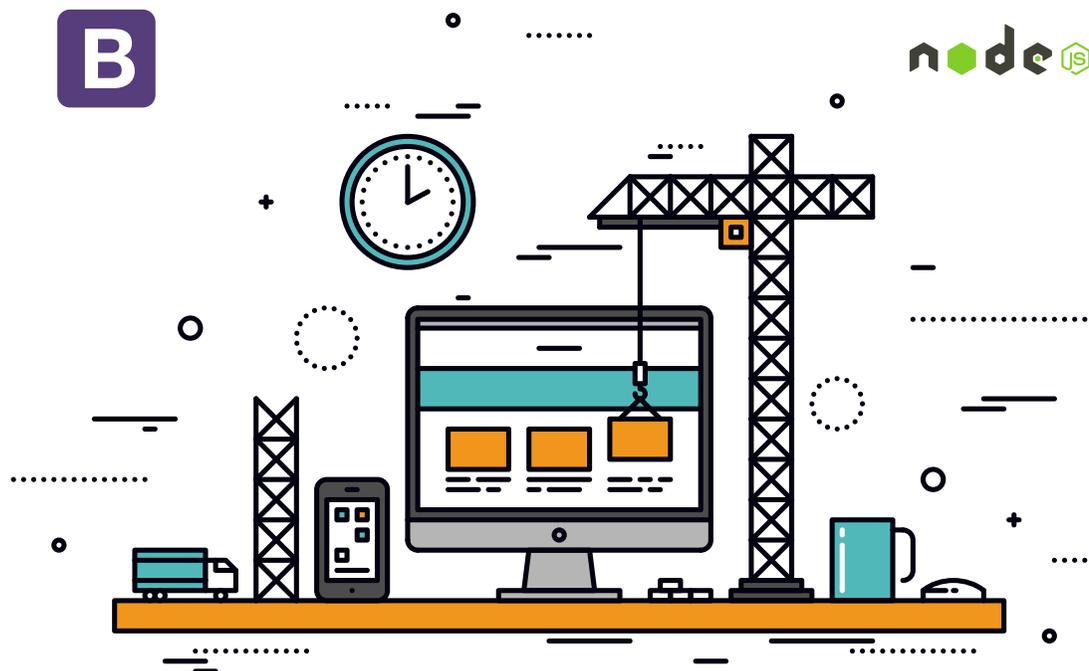


Build a Website Using Bootstrap and the Express.js Framework

The Express.js framework is the most popular and widely used framework for Node.js, while Bootstrap is the frontrunner in the development of responsive, mobile first Web applications on the Net. This article presents a tutorial on how to create a website using the two popular open source frameworks.



Bootstrap is an open source, free and popular front-end framework that's used to design websites and Web applications. It contains HTML and CSS based design templates for buttons, navigation, forms, typography, other interface components and also optional JavaScript extensions. Bootstrap mainly focuses on the front-end development of Web applications.

The Bootstrap framework is also the second most popular project on GitHub, with over 99,000 stars and over 44,000 forks at the time of writing this article. It has some great features, which are explored further in this article.

Browser compatibility: Bootstrap is compatible with all the latest and popular versions of Web browsers, such as Google Chrome, Safari, Opera, Internet Explorer and Mozilla Firefox.

Responsive Web design: Bootstrap version 2.0 onwards has support for responsive Web design, which allows the layout of the Web pages to be adjusted dynamically, depending upon the characteristics of the device used – whether it is a desktop, mobile phone or tablet. From version 3.0 onwards, Bootstrap has adopted a mobile-first-design philosophy, which basically emphasises responsive design, by default.

Style sheets: Bootstrap offers a set of style sheets that

provide basic style definitions for all key HTML elements or components. These basic style definitions provide a uniform, modern appearance for formatting text, tables and form elements.

Reusable components: Bootstrap also contains other commonly used interface elements. These are reusable components that are implemented as CSS classes, which can be applied to certain HTML elements in a Web page.

JavaScript components: Bootstrap has several JavaScript components in the form of jQuery plugins. These JavaScript components provide some additional user interface elements such as alert boxes, tooltips and carousels, and also extend the functionality of some existing interface elements like the auto-complete function for the input fields.

SASS and Flexbox support: The Bootstrap version 4.0 alpha release has SASS and Flexbox support. SASS (Syntactically Awesome Style Sheets) is a scripting language that is interpreted into Cascading Style Sheets. Flexbox, also known as CSS Flex Box Layout, is a CSS3 Web layout model which allows responsive elements within a container to automatically arrange themselves according to different screen sizes and devices.

 **Tip:** The Bootstrap framework is open source and the source code of the framework is available on GitHub. Developers/programmers are encouraged to participate and contribute to the project.

Express.js framework

Express.js or Express is a Web application framework for Node.js designed for building Web applications and APIs. It is the *de facto* standard server framework for Node.js. Express is also the back-end part of the MEAN stack, together with the MongoDB database and the AngularJS front-end framework.

Some of the salient features of the Express.js framework are:

- It's a cross-platform framework, which means it is not limited to one OS.
- It is a server-side Web and mobile application framework written in JavaScript.
- It provides Express Generator, which allows you to create complex applications quickly.
- It supports the MVC pattern.
- Express.js comes with two template engines, Jade and EJS, which facilitate the flow of data into the structure of a website.
- It has a provision for building single-page, multi-page and hybrid Web and mobile applications, as well as APIs or Application Programming Interfaces.

Installing Node.js

Express.js is a Node.js based framework, so Node.js must be installed first. Download and install Node.js (<https://nodejs.org/>). Express.js is distributed as an *npm* package, so we will use Node and *npm* to install Express.js.

Building a website

Let us build a website using these two frameworks – Bootstrap and Express.js.

We will use open source programming languages like HTML, CSS and JavaScript along with these two frameworks, to create a website.

 **Note:** It is assumed that you have a basic knowledge of Web technologies like HTML, CSS and JavaScript. If you don't, W3Schools (<http://www.w3schools.com/>) is a good place to start. The site has some great tutorials for Web technologies which are easy to follow.

First, let's create a folder/directory for our website. Open the command prompt/terminal and type the following command:

```
C:>mkdir project
```

This command creates a folder called *project*:

```
C:>cd project
```

This command will make *project* your working directory. Now, use the *npm init* command to create a very important *package.json* file for the website/ application. Run the following command in the newly created folder, i.e., the *project* folder:

```
C:\project>npm init
```

This command prompts you to enter certain fields, such as name, version of your website or application. You can either hit the *Enter* key and accept the default setting or enter the details you want, with the following exception:

```
Entry point: (index.js)
```

At this point, you will be asked to enter the entry point file name. If you hit *Enter*, it will set the *index.js* file as the entry point, by default. But for this tutorial, we are going to give it a new name. Let's name it *website.js*. Type *website.js* and press the *Enter* key.

Now, let us install Express in the *project* directory and save it in the dependencies list. Run the following command in the *project* folder:

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

The above command will install all the important Express modules, and save the dependencies in the *package.json* file.

The *package.json* file will look like what follows:

```
{
  "name": "project",
  "version": "1.0.0",
  "description": "Sample Project",
  "main": "website.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "AniketKudale",
  "license": "ISC"
}
```

Here's a description of the code snippet given above.

- *name*: Name of the Web application.
- *version*: Version of the Web application.
- *description*: Description of the Web application.
- *main*: Main entry point of the Web application; in this file, we store our application logic.
- *scripts*: Here, we specify script commands that run at various intervals in the life cycle of our package.
- *author*: Here, we specify the author's name, i.e., your name.
- *license*: Here, we can specify the licence type.

The Express.js framework has now been successfully

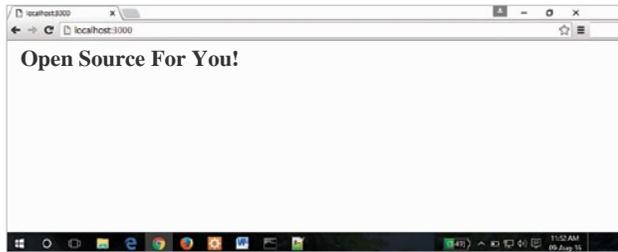


Figure 1: Sample example running in the browser

installed; so let us test it by creating a sample ‘Hello World’ example.

In our project directory, create a file named *website.js* and add the following code:

```
var express = require('express');
var app = express();
app.get('/', function (req, res) {
  res.send('<h1>Open Source For You!</h1>');
});
app.listen(3000, function () {
  console.log('Example app listening on port 3000!');
});
```

The above code starts a server and listens on port 3000 for connections. It also responds with the text ‘Open Source For You!’ in the header 1 HTML formatting for requests made to the root URL (/), and for every other path, it will respond with a message ‘404 Not Found’.

To execute, run the following command in the *project* folder:

```
C:\project>node website.js
```

Then, load *http://localhost:3000/* in the browser to see the output.

If you see the text printed in the browser, as shown in Figure 1, then you are all set for creating a website using Bootstrap and Express.js.

Creating a website

We are going to use Bootstrap and basic HTML for the view, and the Express.js framework as a Web server and to handle routes.

You can download all the necessary Bootstrap files from *http://getbootstrap.com/* or you can use files from CDN (Content Delivery Network).

For our sample website, we are going to use Bootstrap files from CDN.

Let’s start by creating views first; our website will have three Web pages.

1. *index.html*: Index page of our sample website.
2. *product.html*: The product page of our sample website, where we will add some sample information about products.
3. *about.html*: The ‘About us’ page of our sample website, where we will add contact details, etc.

Creating the HTML pages

Navigate to our *project* folder, and create a new folder called ‘views’ in it. After that, open Notepad or any of your favourite code editors, and copy the following HTML code:

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

<head>
<meta charset="UTF-8">
<title>Sample website using Bootstrap and ExpressJS</title>

<!--CDN Links-->
<script src="//ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js"></script>
<link rel="stylesheet" href="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.1/css/bootstrap.min.css">
<script src="//maxcdn.bootstrapcdn.com/bootstrap/3.3.1/js/bootstrap.min.js"></script>

<script src="website.js"></script>
</head>
<body>
<div>
<div>
<nav class="navbar navbar-inverse" role="navigation" style="padding-left:130px;">
  <ul class="nav navbar-nav">
    <li class="active"><a href="/">Home<span class="sr-only">(current)</span></a></li>
    <li><a href="/product">Products</a></li>
    <li><a href="/about">About us</a></li>
  </ul>
</nav>
</div>
<br/>
<div class="jumbotron"> <p>
This is place to put your Sample Content.
</p></div>
</div>
</body>
</html>
```

Save the above code as *index.html* in the *views* folder, which is present inside the folder named *project* (i.e., at *C:\project\views*).

As you can see in the above code, we have used Bootstrap and jQuery files from CDN. Also, we have included the *website.js* file, where we are going to write routing logic for this sample website. In the above code, we have also used Bootstrap’s Navbar class to provide navigation to the HTML pages present in the *views* folder. Since this is the index page, we have set the class of the *Home* link in the navbar as ‘active’.

For *product.html* also, open Notepad or any of your favourite code editors and copy the following HTML code:

```
<html>
<head>
<link rel="stylesheet" href="http://maxcdn.bootstrapcdn.com/
bootstrap/3.3.1/css/bootstrap.min.css">
<script src="//maxcdn.bootstrapcdn.com/bootstrap/3.3.1/js/
bootstrap.min.js"></script>
</head>
<body>
  <div>
    <div>
      <nav class="navbar navbar-inverse" role="navigation"
style="padding-left:130px;">
        <ul class="nav navbar-nav">
          <li><a href="/">Home</a></li>
          <li class="active"><a href="/product">Products<span
class="sr-only">(current)</span></a></li>
          <li><a href="/about">About Us</a></li>
        </ul>
      </nav>
    </div>
    <br/>
    <div class="jumbotron">
      <p>
        Put the product details here!
      </p>
    </div>
  </div>
</body>
</html>
```

As you can see in the above code too, we have used Bootstrap files from CDN, and we have also used Bootstrap's Navbar class to provide navigation to HTML pages present in the *views* folder. In this case, since this is the product page, we have set the class of the *Products* link in Navbar as 'active'.

Similarly, for *about.html*, open Notepad or any of your favourite code editors and then copy the following HTML code:

```
<html>
<head>
<link rel="stylesheet" href="http://maxcdn.bootstrapcdn.com/
bootstrap/3.3.1/css/bootstrap.min.css">
<script src="//maxcdn.bootstrapcdn.com/bootstrap/3.3.1/js/
bootstrap.min.js"></script>
</head>
<body>
  <div>
    <div>
```

```
<nav class="navbar navbar-inverse" role="navigation"
style="padding-left:130px;">
  <ul class="nav navbar-nav">
    <li><a href="/">Home</a></li>
    <li><a href="/product">Products</a></li>
    <li class="active"><a href="/about">About Us<span
class="sr-only">(current)</span></a></li>
  </ul>
</nav>
</div>
<br/>
<div class="jumbotron">
  <p>
    Put the contact details here!
  </p>
</div>
</div>
</body>
</html>
```

Here, too, we have used Bootstrap files from CDN, and we have used Bootstrap's Navbar class to provide navigation to HTML pages present in the *views* folder. In this case, since this is the *About* page, we have set the class of the 'About Us' link in the Navbar as 'active'.

We are now done with the view/presentation part. Let's add logic to these Web pages by making use of the Express.js framework.

Let's navigate to the root of our *project* directory, open the file *website.js* present there, and delete the code inside it before copying the following code:

```
var express = require('express');
var app = express();
var router = express.Router();

var path = __dirname + '/views/';

app.use('/', router);

router.get('/', function(req, res){
  res.sendFile(path + 'index.html');
});

router.get('/product', function(req, res){
  res.sendFile(path + 'product.html');
});

router.get('/about', function(req, res){
  res.sendFile(path + 'about.html');
});
```

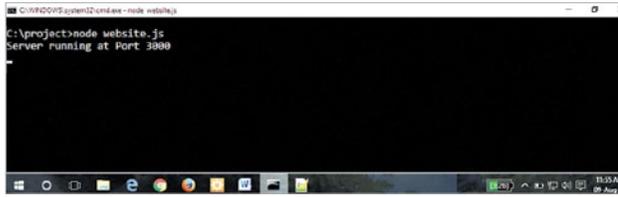


Figure 2: Server running and listening for connections on a port

```
app.use('*',function(req, res){
  res.send('Error 404: Not Found!');
});

app.listen(3000,function(){
  console.log("Server running at Port 3000");
});
```

Here is an explanation of the code snippet given above.

First, we load the dependencies, i.e., the Express.js framework. We are also loading the `router()`, which is the built-in routing service provided by the Express.js framework.

Since we have stored our HTML files in the 'views' folder, we have assigned the path using the `__dirname` keyword, which basically points to our current working directory.

Then, we used `app.use('/', router)` since we are using routes in the code.

After this, we define the routes: `/product` and `/about`. These router definitions have a `sendFile()` function, which is a built-in function and is designed to send files to the Web browser. For example, in our case, in the `index.html` page, if the user clicks on any one of the Navbar links, then the `router.get()` function provides the file associated with that particular link.

And if the user enters some invalid routes, we can also display the custom error message by using the `* regex` pattern in the `app.use()` function.

Finally, we declare the port number that listens to connections using the `app.listen()` function.

Your `project` folder must end up with the following directory structure:

```
-- node_modules
-- views
+ -- index.html
+ -- product.html
```

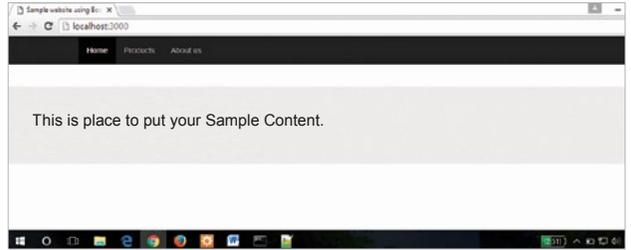


Figure 3: The website we created, running in the browser

```
+ -- about.html
package.json
website.js
```

The above `project` directory structure has two folders (named `node_modules` and `views`) and two files (named `package.json` and `website.js`). The `views` folder contains three HTML files named `index.html`, `product.html` and `about.html`.

Running the website

To execute, run the following command in the project folder:

```
C:\project>node website.js
```

The message shown in Figure 2 will be displayed.

Then, load the address `http://localhost:3000/` in the browser to see the output.

You should be able to see your first website created using Bootstrap and the Express.js framework, as shown in Figure 3. **END** 🐧

References

- [1] <http://getbootstrap.com/>
- [2] <https://expressjs.com/>
- [3] <https://en.wikipedia.org/wiki/Express.js>
- [4] <http://www.w3schools.com/bootstrap/default.asp>

By: Aniket Eknath Kudale

The author is an open source enthusiast who has more than two years of experience as a software engineer at Tibco Software Inc., Pune. You can reach him at kudale@aniket.co

Know the Leading Players in Every Sector of the Electronics Industry

www.electronicb2b.com

**ACCESS ELECTRONICS
B2B INDUSTRY WITH A**

CLICK OF A BUTTON

Log on to www.electronicb2b.com and be in touch with the Electronics B2B Fraternity 24x7