Building a Site with Node and Express

[Node.js (http://nodejs.org/)](http://nodejs.org/) has grown increasingly in popularity over the past couple years. With its adoption by large companies like Microsoft, Yahoo, PayPal, eBay, and many more, now is a great time to jump into Node development.

What We'll Be Building

In this booklet, we'll be looking at how to create a simple 3-page website using Node and its most popular framework, [ExpressJS (http://expressjs.com/)](http://expressjs.com/).

Here's a picture of the site we'll be creating. Nothing fancy from a design perspective. Our main focus will be on the Node and Express side of things and we'll just use [Twitter Bootstrap (http://getbootstrap.com/)](http://getbootstrap.com/) for quick styling.
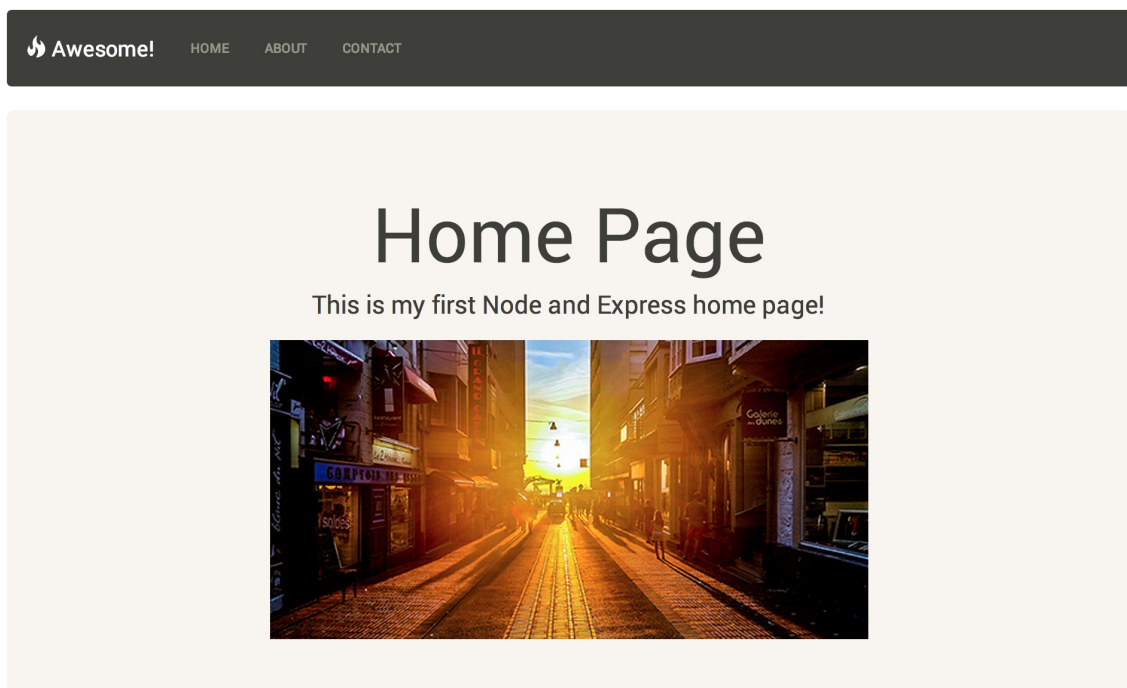
By building a sample site using Node and Express, we will learn many things including:

- Node concepts, best practices, and getting started
- Express concepts, best practices, and getting started
- Routing applications with Express
- How to use [EJS (http://embeddedjs.com/)](http://embeddedjs.com/) (a JavaScript templating engine) to template views
- How to pass data and variables from server to HTML

We'll have 3 pages with 2 different layout types:

- Full Width Page (**Home** and **Contact**)

- Page with Sidebar (**About**)

By using a full page and a sidebar layout, we'll be able to see how we can template our views. This will benefit us because we don't have to rewrite our view files over and over. [DRY (http://en.wikipedia.org/wiki/Don't_repeat_yourself)](http://en.wikipedia.org/wiki/Don't_repeat_yourself) is the way to go!

Now that we know what we are building, lets get started with the fun stuff, the actual programming!

Starting our Application

Let's start out by looking at the file structure for our application. This is a good way to get a top-down view and now what files we will need. Here are the files we have:

- public (folder that will hold css/js/images)
- views (will have our view files)

    - partials (the repeatable things for our site (head, header, footer))
    - pages (the main pages for our site (home, about, contact))

- package.json (where we start our Node/Express application)
- server.js (where we configure Express and define site routes)

When starting a Node application, we will always start with the `package.json` file. This is where we define the main parts of our application like its name, version, author, license, and dependencies.

Let's create our `package.json` file with the minimal attributes needed to start our application.

```
{
    "name": "node-express-site",
    "main": "server.js",
    "dependencies": {
        "express": "~4.8.5",
        "ejs": "~1.0.0"
    }
}
```

Shortcut for Creating a package.json File: If you want an easy way to create the `package.json` file, npm comes with a great starting command: `npm init` . Just type that and watch the magic as your package.json file is generated for you.

Shortcut for Adding Dependencies: When adding dependencies, you won't always know the version number of the packages that you want. npm comes with another shortcut for adding dependencies to your project. Just type `npm install <package name> --save` . npm will automatically add your package to the dependencies section with the latest version!
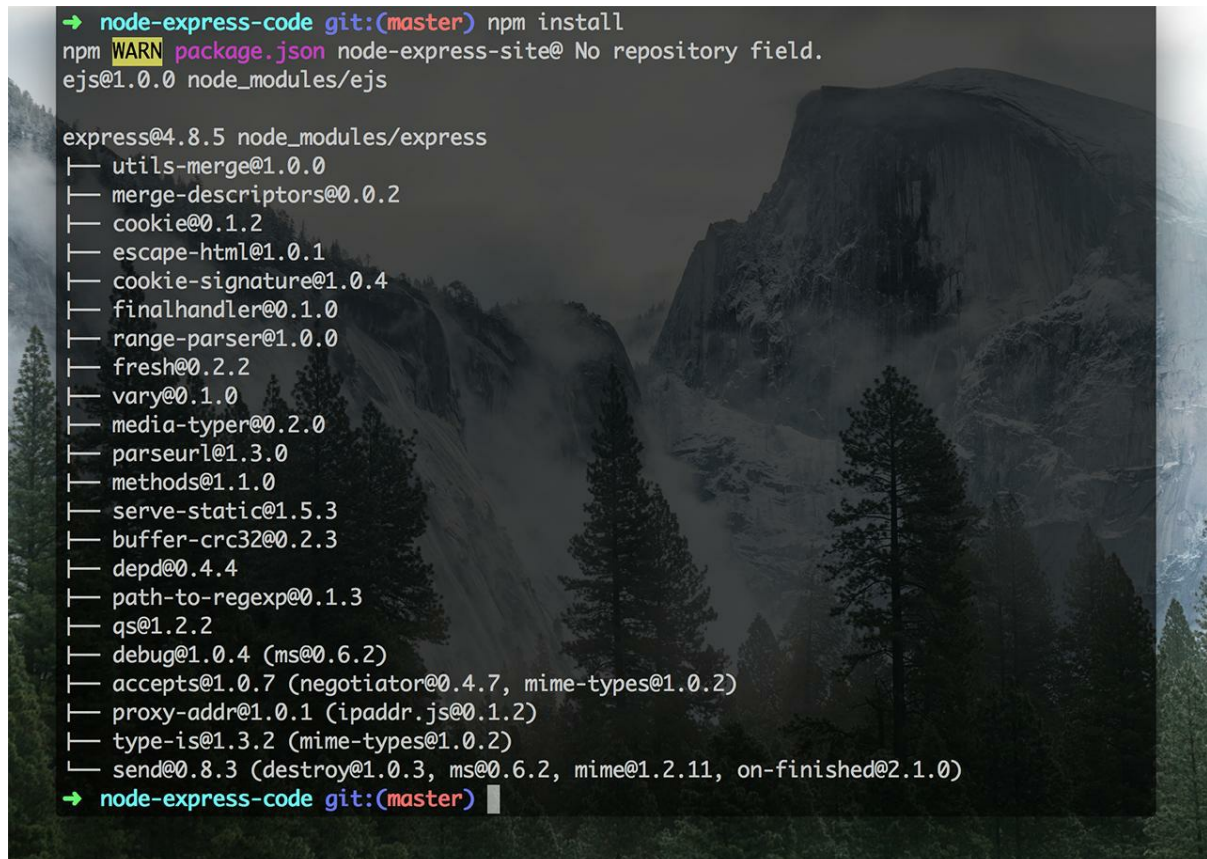
Installing Express and EJS

In the above `package.json` file, we have defined:

- **name**: The name of our application
- **main**: The file that we will use to start up our application.
- **dependencies**: The dependencies that we will need (Express and EJS).

By adding `express`, `ejs`, and the version we want to the dependencies, we can now bring in both of these packages by running:

```
npm install
```


We can see npm bring in the express and the ejs packages and place them into the **node_modules** folder that gets created.

Now we have **defined our application** and **have the dependencies we need**. Let's start configuring our application using Node and Express in our `server.js` file.

Starting a Node and Express Server

We defined our main file earlier in our `package.json` file. We will be using a file called `server.js`. In this file, we will:

- Set up a Node server using Express
    - We will be able to visit our site in our browser at `http://localhost:8080`

- Configure our app to **use ejs** as the templating engine
- Set up our routes
- Start the server!

Let's start up our `server.js` file and break it down for each section. We'll start by calling Express.

```
// CONFIGURATION ====================
// ==================================
// load the express and create our application
var express = require('express');
var app     = express();

// set the port based on environment
var port    = process.env.PORT || 8080;

// START THE SERVER ================
// ==================================
app.listen(port);
console.log(port + ' is the magic port!');
```

In this block of code, we are grabbing express, creating the application, and setting our port. All of this will create a Node server so that we can visit our application in browser.

Defining the `port` using `process.env.PORT` lets us dynamically set the port based on environment. This is useful when deploying our application to hosting environments like Modulus (https://modulus.io/) since our application will be set to the correct port when starting up.

By using `app.listen(port)`, our server will be started. Let's go into the command line and type:

`node server.js`
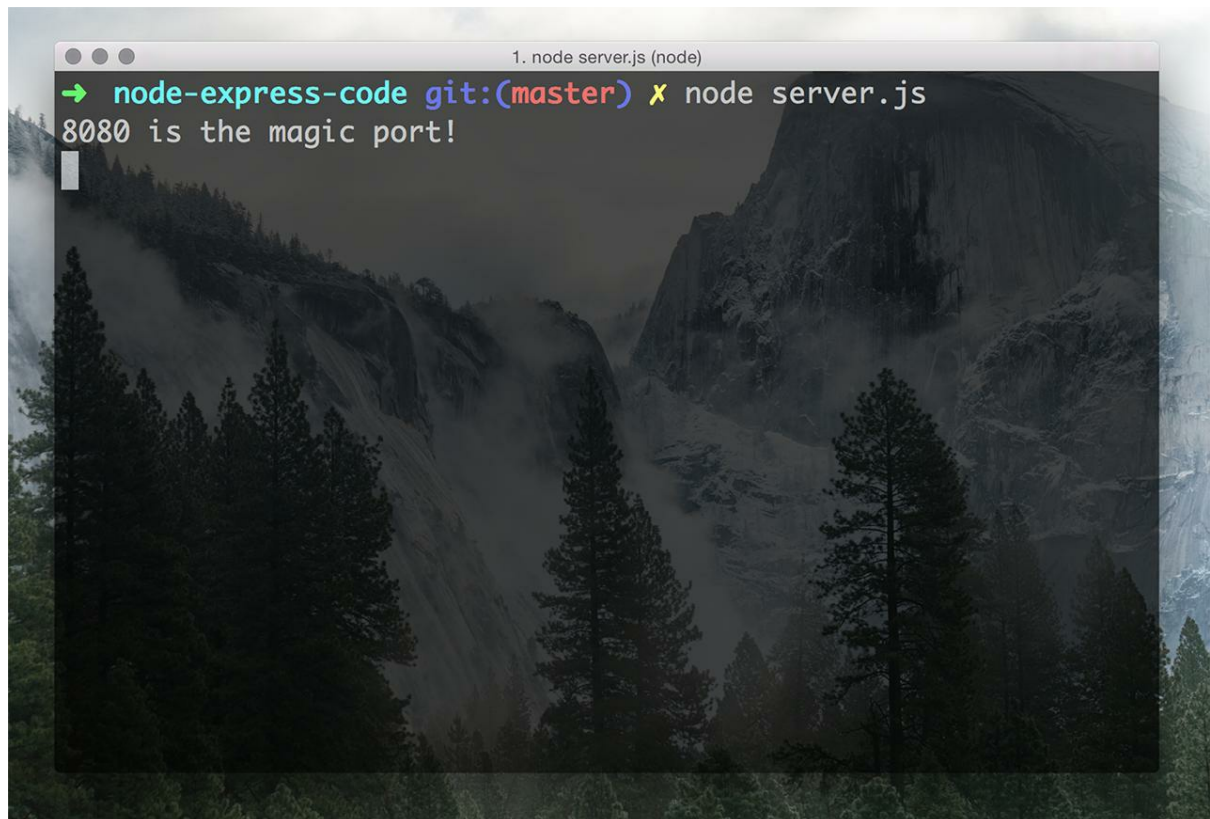
Automatically Restarting Your Node Server. When we use `node server.js` , we start up our server. The problem with this way of starting the server during development is that we will need to shut the server down and then restart it to see new changes every time we update our files. This could be a tedious when making constant file changes. Luckily there is a package that we can install that will automatically restart the server on file changes. Say hello to a package called **nodemon**. Install it using `npm install -g nodemon` and then you can start your server with `nodemon server.js` . Watch the magic happen when you update your files and your server restarts itself!

Our server is now up and running and since we set our port to `8080` , we are now able to visit the site in our browser at `http://localhost:8080` .

Unfortunately, we won't see anything in our browser yet because we haven't defined any routes or data to show our users yet. Let's get to that now.

Setting Up Express Routes

When creating routes, Express comes with its [Router (http://expressjs.com/api#router)](http://expressjs.com/api#router). We can use this to create routes on the `app` object we made earlier. Here is a very simple route to send a string to our users in the browser.

```
// index/home page
app.get('/', function(req, res) {
    res.send('Look! I am the home page!');
});
```

The `res.send` command will send this string to our user. This is a limited command since we are only sending a little sentence to our user and not sending a full web page to our users. Hang tight though, we'll get to the HTML and CSS soon.

Now once we start up our server using `node server.js` (or `nodemon server.js`), we can finally go into our browser and see our application at `http://localhost:8080`.

PICTURE HERE

Let's add the other 2 routes that we will need for our other pages (**About** and **Contact**). Add these to your `server.js` file.

```
// about page
app.get('/about', function(req, res) {
    res.send('Hey there! I am the about page');
});

// contact page
app.get('/contact', function(req, res) {
    res.send('Looking to contact someone?');
});
```

We can see both of these in our browser now. Up to this point, we have successfully **created our Node and Express server**, **set up our application routes**, and **sent data to our users**. While this is impressive to us, our users will probably want to see more than just a message.

The next step is to set up a good looking HTML/CSS site and send that to our users. Then we'll have a fully functioning website to show off!

Using EJS as our View Engine

EJS (http://embeddedjs.com/) is one of the templating engines that we can use with Express. Some other templating engines we can use are Jade, Haml, hbs (handlebars), and swig. Take a look at the list of templating engines (https://github.com/strongloop/express/wiki#template-engines) that are available.

We'll use EJS since the syntax is very easy to use and sticks close to standard HTML syntax and standards.

Since we already took care of installing EJS earlier in our `package.json` file, we just need to configure our application to use it. Add this line to our `server.js` file and we have turned on EJS as our templating engine.

```
// set the view engine to ejs
app.set('view engine', 'ejs');
```

Easy stuff! Let's move onto our view files.

Setting Up The Views

Earlier when we looked at the file structure, we had a **views** folder. Go ahead and create a folder inside of that called **pages** and create a file called `home.ejs` . This will be the file for our home page and we'll send this to our users inside of our `app.get('/', ...` route.

We'll keep this file very simple and use Bootstrap classes to style our page. We'll load a Bootstrap file from [Bootstrap CDN (http://www.bootstrapcdn.com/)](http://www.bootstrapcdn.com/). Feel free to use the default Bootstrap file, any of the Bootswatch files, or just not use Bootstrap and style things fully custom.

Here's our `home.ejs` file in all its glory.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Node and Express Site</title>

    <!-- CSS -->
    <!-- load a bootstrap theme called sandstone -->
    <!-- referencing local files will look in the /public folder -->
    <link rel="stylesheet"
href="//maxcdn.bootstrapcdn.com/bootswatch/3.2.0/sandstone/bootstrap.min.css">
</head>
<body class="container">

    <header>
        <nav class="navbar navbar-default" role="navigation">

            <div class="navbar-header">
                <a class="navbar-brand" href="#"><span class="glyphicon glyphicon-fire"></span> Awesome!
</a>
            </div>

            <ul class="nav navbar-nav">
                <li><a href="/">Home</a></li>
                <li><a href="/about">About</a></li>
                <li><a href="/contact">Contact</a></li>
            </ul>

        </nav>
    </header>

    <main>
        <div class="jumbotron text-center">
            <h1>Home Page</h1>
            <p>This is my first Node and Express home page!</p>
        </div>
    </main>
```

```
    <footer>
        <p class="text-center text-muted">
            Copyright &copy; 2014 Cool Programmers
        </p>
    </footer>

</body>
</html>
```

A lot of this is standard HTML using the Bootstrap classes. EJS is easy to use since our files look exactly like any other HTML site we would create.

Sending a View to Our User

We've created our EJS file, and now we have to send this to our users. Update your home page route in `server.js` to send back the newly created view:

```
// index/home page
app.get('/', function(req, res) {
    res.render('pages/home');
});
```

Express let's use send an EJS template file to our users using `res.render()`.

File Structure for EJS View Files: By default, Express and EJS will look in a **views** folder in the root of your application so make sure you reference files from that folder.

Load up your server again using `node server.js` (or `nodemon server.js`) and visit your site at `http://localhost:8080`.

PICTURE HERE

Great stuff! Let's go ahead and update our other routes so that we are using `res.render()` again.

```
// about page
app.get('/about', function(req, res) {
    res.render('pages/about');
});

// contact page
app.get('/contact', function(req, res) {
    res.render('pages/contact');
});
```

We have a good looking home page now, but let's look to improve it by adding some custom css and

Adding Public Assets (CSS/JS/Images)

We want this site to become the best site that it can possibly be so let's add in some custom CSS and images.

For our Express site, we will want to access css from our site. This means we have to make sure our application knows where to find our resources. We already created the **public** folder earlier, so we only have to point Express to that directory when calling assets. This can be done with one line in our `server.js` file.

```
// set the path for all public resources (css/js/images)
app.use(express.static(__dirname + '/public'));
```

Next, we will create a css file that we can put all of our custom css at. Create the following file: `public/css/style.css`

```
body    { background:#DDD; padding-top:50px; }
```

Now that we have told Express where to look for assets and created the asset, the last order of business is to add it to our site.

Add the following line to your `home.ejs` file in the `<head>` of the document.

```
<link rel="stylesheet" href="css/style.css">
```

Our application will automatically look in the **public** folder for this stylesheet and when we refresh our site, we have our stylesheet changes!

PICTURE HERE

Let's also add a picture just so we can see how we can easily add pictures. Take any photo you want and add it to a **public/img** folder. Now you can go into your `home.ejs` file and add that like you would normally:

```
<main>
    <div class="jumbotron text-center">
        <h1>Home Page</h1>
        <p>This is my first Node and Express home page!</p>

        <img src="img/city.jpg" alt="This is the city!">
    </div>
</main>
```

Our site will go into the **public** folder when looking for that image and display add it to our site!

Awesome!　HOME　ABOUT　CONTACT

Home Page

This is my first Node and Express home page!

We have now **created an ejs file**, **displayed that to our user** as the home page, **routed public assets**, and **added a css file and an image**.

Let's move onto the other two pages of our site.

Templating Our Application

When creating the other two pages of our site, we won't want to repeat the entire page code from `home.ejs` over and over. The repeatable code are things like the `<head>` of the document, the `<header>`, and the `<footer>`.

We're going to pull that code out of the `home.ejs` file and store those into **partials**. By doing this, we can include these files into each of our pages and be efficient and DRY with our code.

Inside of the **views/partials** folder create 4 different EJS files: `head.ejs`, `header.ejs`, `footer.ejs`, and `sidebar.ejs` (we'll use this for the About page).

We're going to take the respective parts outside of `home.ejs` and use EJS includes instead.

Here is the code for each of those 4 files we just created:

`views/partials/head.ejs`

```
<meta charset="UTF-8">
<title>Node and Express Site</title>
```

```
<!-- CSS -->
<!-- load our css file and a bootstrap theme called paper -->
<!-- referencing local files will look in the /public folder -->
<link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootswatch/3.2.0/sandstone/bootstrap.min.css">
<link rel="stylesheet" href="css/style.css">
```

views/partials/header.ejs

```
<nav class="navbar navbar-default" role="navigation">

    <div class="navbar-header">
        <a class="navbar-brand" href="#"><span class="glyphicon glyphicon-fire"></span> Awesome!</a>
    </div>

    <ul class="nav navbar-nav">
        <li><a href="/">Home</a></li>
        <li><a href="/about">About</a></li>
        <li><a href="/contact">Contact</a></li>
    </ul>

</nav>
```

views/partials/footer.ejs

```
<p class="text-center text-muted">
    Copyright &copy; 2014 Cool Programmers
</p>
```

views/partials/sidebar.ejs

```
<div class="panel panel-primary">

    <div class="panel-heading">
        <h3 class="panel-title">Popular Stuff</h3>
    </div>

    <div class="panel-body">

        <div class="list-group">
            <a href="#" class="list-group-item">The Sun is Gigantic</a>
            <a href="#" class="list-group-item">Top 10 Fails</a>
            <a href="#" class="list-group-item">Coffee is Important</a>
            <a href="#" class="list-group-item">Programming in Node is Too Fun</a>
        </div>

    </div>
</div>
```

Inserting Partials with EJS

Now that we have defined our **partials files**, we need to include them in our **pages files**. To include a partial using EJS, use the following:

```
<% include file_name %>
```

When including a file like this, the path to that file is **relative to the current file**. Let's use home.ejs as the main example.

Home Page

Here is `home.ejs` using includes:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <% include ../partials/head %>
</head>
<body class="container">

    <header>
        <% include ../partials/header %>
    </header>

    <main>
        <div class="jumbotron text-center">
            <h1>Home Page</h1>
            <p>This is my first Node and Express home page!</p>

            <img src="img/city.jpg" alt="This is the city!">
        </div>
    </main>

    <footer>
        <% include ../partials/footer %>
    </footer>

</body>
</html>
```

Contact Page

Let's also create our **Contact Page** (`views/pages/contact.ejs`). This page will use a full width layout so it is very similar to our home page. It just uses different content in the `<main>` section and thanks to our includes, we don't have to repeat a lot of code!

Here's the code for `contact.ejs` :

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
    <% include ../partials/head %>
</head>
<body class="container">

    <header>
        <% include ../partials/header %>
    </header>

    <main>
        <div class="jumbotron text-center">
            <h1>Contact Page</h1>
            <p>Fill out the contact form if you can get past the bear!</p>

            <img src="img/bear.jpg" alt="Bear in Your Face">
        </div>
    </main>

    <footer>
        <% include ../partials/footer %>
    </footer>

</body>
</html>
```

Here is our new contact page viewed in the browser at `http://localhost:8080/contact`:

PICTURE HERE

About Page (with a sidebar)

The last page we have to create is our About Page. This page will differ from the other two pages because it will have a sidebar attached to it. We'll use Bootstrap's grid classes (`col-sm-8` and `col-sm-4`) to define a main part of the page and the sidebar.

Here's the code for the `about.ejs` file:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <% include ../partials/head %>
</head>
<body class="container">

    <header>
        <% include ../partials/header %>
    </header>

    <main>
```

```
        <div class="row">

            <div class="col-sm-8">
                <div class="jumbotron text-center">
                    <h1>About Page</h1>
                    <p>This is my first Node and Express home page!</p>

                    <img src="img/raspberries.jpg" alt="Food!">
                </div>
            </div>

            <div class="col-sm-4">
                <% include ../partials/sidebar %>
            </div>

        </div>

    </main>

    <footer>
        <% include ../partials/footer %>
    </footer>

</body>
</html>
```

We are using other Bootstrap classes here like the `panel` and the `list-group` . Give the [Bootstrap docs (http://getbootstrap.com/components/)](http://getbootstrap.com/components/) a look through to see the cool prebuilt classes they give you access to.

PICTURE HERE

Now we have our **About Page with a sidebar**! By using **ejs includes**, we can quickly and easily create multiple types of page types.

Passing Data to Our Views

The last part of our site that we will deal with is passing dynamic data to our views. In the future, we will want our site to be dynamic and have the ability to pull in data from a database or multiple sources online via APIs.

While getting dynamic data is outside of the scope of our tutorial, we'll look at how we can take data and pass it into our views using EJS.

Passing Data in Each Route

Data is passed to each view in an Express route inside the `res.render()` function.

We are going to pass the following data into each view:

- A variable into the home page
- A list into the about page
- A boolean variable into the contact page

Here are the updated routes in our `server.js` file with data passed into each:

```javascript
// index/home page
app.get('/', function(req, res) {
    res.render('./pages/home', {
        name: 'John Doe'
    });
});

// about page
app.get('/about', function(req, res) {
    // define a list of people
    var people = [
        { name: 'Bruce Wayne', alias: 'Batman' },
        { name: 'Clark Kent', alias: 'Superman' },
        { name: 'Scott Lang', alias: 'Ant Man' }
    ];

    res.render('pages/about', {
        people: people
    });
});

// contact page
app.get('/contact', function(req, res) {
    res.render('pages/contact', {
        active: true
    });
});
```

Each route now has data being passed to it. Let's go one by one and use that data in each respective view.

Use EJS to Display a Variable (Home Page)

PICTURE HERE

Use EJS to Display a List (About Page)

PICTURE HERE

Use EJS for an if Statement (Contact Page)

PICTURE HERE

If you go back into `server.js` and change `active` to **false** and refresh that contact page, the message

will disappear.

Conclusion and Further Reading

We made it! A fully functioning and good looking website using Node, Express, and EJS!

To recap all the things we've learned:

- 1
- 2
- 3