Build your first Node.js website, Part 2

Ruth Willenborg

February 25, 2015
(First published July 29, 2014)

Do you want to build a dynamic website with Node.js but are unsure where to start? This three-part article introduces you to Node.js development step by step, with no software installation required. Using only a web browser, you create a Node.js application, write server-side JavaScript, display dynamic data on HTML pages, and store data in a Redis database.

To view the introductory video **Build your first Node.js website, Part 2** , please access the online version of the article. If this article is in the developerWorks archives, the video is no longer accessible.

This is the second part of a three-part article that introduces you to Node.js development step-by-step, with no software installation required.

Part 1's "Hello World"-plus application was a great starting point. You used IBM Bluemix DevOps Services and IBM Bluemix™ to bring up a Node.js server, and you coded some simple server-side JavaScript to count the requests to your server. Here in Part 2, you'll add formatted HTML pages to your site and then pass the data from the server for display on the home HTML page.

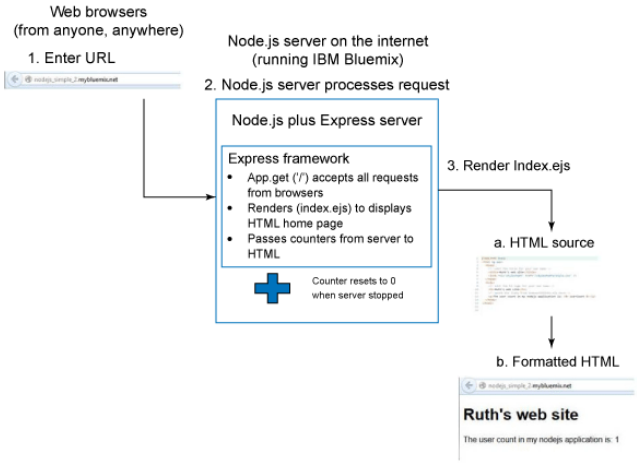
To add a web front end to Node.js, the most common technique is to use the Express web application framework. With Express, you have several choices for formatting web pages, including Jade, Handlebars, and EJS. To stay in my comfort zone, I chose EJS, which uses plain old HTML.

“ The app.js file includes all the Express setup. ”

This part starts with a new Bluemix DevOps Services project, so it is not necessary to complete the exercise in Part 1 before proceeding.

I found a good Express EJS starting sample, stripped it down to the bare minimum, and then extended it to pass data and serve static web pages. You'll start with my version and learn how to display data from the Node.js application in an HTML page and how to serve up static HTML content. In Part 3, you'll use a Redis database to persist data for the website.

The application's modified flow now looks like this:



What you'll need

- A Bluemix account.

Get the code

Step 1: Create and deploy your copy of the HTML application

1. Log in to Bluemix DevOps Services.
2. Click this article's **Get the code** button above.
3. In Bluemix DevOps Services, click the **FORK PROJECT** button on the menu to create your own copy of the code, provide a unique name, and click the **CREATE** button.
   Now you have the directory structures and file setup for Express. The app.js file includes all the Express setup:

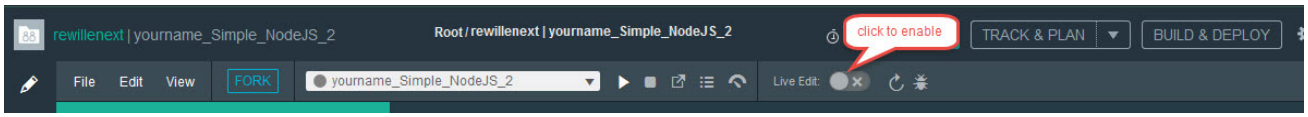
```
var app = express();
app.set('port', process.env.PORT || 3000);
app.set('view engine', 'ejs');

app.use(morgan('dev'));
app.use(express.static(path.join(__dirname, 'public')));
```

   The package.json file is also updated to add dependencies for Express and EJS:

```
"dependencies": {
    "express": "*",
    "ejs": "*",
    "morgan": "*"
  }
}
```

   Click the **EDIT CODE** button to begin working with the application.
4. Enable Live Edit and click **OK** to redeploy the application:



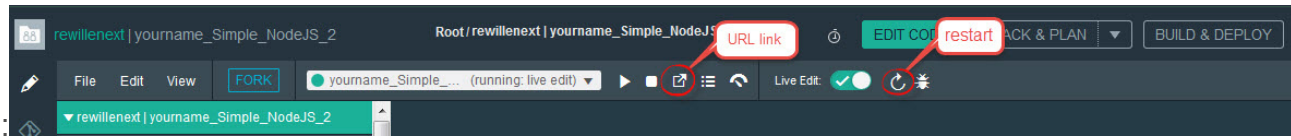
5. Click the URL link in the runbar to open the formatted HMTL page in a new browser tab.
6. Refresh the page and watch the counter increment with each refresh.

Step 2: Change the HTML index page and redeploy

> In app.js, the Express setup specifies `app.set('views', path.join(__dirname, 'views'));`. This statement tells the application to look in the views directory for the initial page to render.

Now you'll make some minor changes to the HTML index page:

1. Back in the Bluemix DevOps Services IDE, open the views folder and click index.ejs to edit the HTML.
2. Edit the text in the `<title>` and `<h1>` tags to replace Ruth with your own name.
3. Restart your application using Live Edit Restart, and click the URL link in the runbar to open the application. Verify that the changes are displayed:



Step 3: Pass server-side content to the HTML page

It's time to learn a little of the Express EJS magic. To display data from your Node.js application onto an HTML page, you must pass the data to the web page using this syntax:

```
res.render('index', {userCount: userCount})
```

`index` is the name of the initial HTML page displayed. Inside the `{}` are the names of the variables to pass in, along with their values.

Make these changes to the application:

1. Back in the Bluemix DevOps Services IDE, select app.js. Add a new `userbytwo` variable and initialize it:
   ```
   var userbytwo = 0;
   ```
2. Increment `userbytwo`:
   ```
   userbytwo = userbytwo + 2;
   ```
3. Update the `render` command to add `userbytwo` as a variable to pass to the HTML page for display:
   ```
   res.render('index', {userCount: userCount, userbytwo: userbytwo});
   ```

   Your `app.get` code block should now look like this:
   ```
   var userCount = 0;
   var userbytwo = 0; /* added var definition for userbytwo here */

   app.get('/', function(req, res){
     userCount = userCount + 1;
     /* add statement to increment userbytwo by two here */
     userbytwo = userbytwo + 2;
     res.render('index', {userCount: userCount, userbytwo: userbytwo});
    /* updated this line */
   });
   ```

Step 4: Display variables on an HTML page

Naturally, the HTML page also needs some special syntax to retrieve and display the passed variables:

1. Return to the views folder and select index.ejs.
2. Copy the <p> line that displays `userCount` and paste it immediately below the existing line. In the new line, change `userCount` to `userbytwo` to display both variables:
```
<p>The user count in my nodejs application is: <%= userCount %></p>
<p>The user by two count in my nodejs application is: <%= userbytwo %></p>
```
3. Restart your application using Live Edit Restart, and click the URL link in the runbar to open the application. Verify that the changes are displayed.

Step 5: Add linked HTML pages

Now you'll add a few links to your website's index page. Linking to additional pages isn't hard, but you must ensure that the directory structures are set up correctly to find your pages.

In the Express setup in app.js, the `app.use(express.static(path.join(__dirname, 'public')));` statement tells the application to look in the public directory for HTML content. To get you started, the public folder contains two HTML pages, with images and stylesheets subdirectories.

1. Open the views folder, select indexwithlinks.ejs, and copy lines 9-15.
2. Open index.ejs, paste the copied content after line 12, and save. You can see that the copied lines link to sports.html and aboutme.html.
3. Open the public directory and locate aboutme.html and sports.html.
4. Make any changes you'd like to aboutme.html and sports.html. You can even upload your own picture into the images directory and link to it.
5. Restart your application using Live Edit Restart, and click the URL link in the runbar to open the application. Verify that the changes are displayed.

Conclusion to Part 2

Here in the second part of this three-part article, you created a multipage website with dynamic, server-side data displayed from your Node.js application. As in Part 1, the entire example was done with just a web browser accessing Bluemix DevOps Services and Bluemix— no installation required.

Now you're ready to move on to Part 3 and store the data in a database.

Related topics

- See IBM Bluemix in action
- Develop and deploy a Node.js app
- My First Website CoderDojo HTML project
- The SDK for Node.js runtime
- Node.js
- JavaScript