**HTML Templating Within Express.js -** [Ryan Schleck](https://ryan-m-schleck.medium.com/?source=post_page-----7b668e34482a--------------------------------) Jul 3, 2022

**Introduction**

As far as web frameworks go, [Express.js](https://expressjs.com/) is a very popular option. As I’ve written about in [a previous blog post](https://ryan-m-schleck.medium.com/using-express-for-static-html-bc4235351473), Express.js can easily be used to serve static html pages. Writing all of this html by hand however can become quite cumbersome as well as limiting in what you can display. These days displaying html content in a dynamic way using an html [templating](https://en.wikipedia.org/wiki/Web_template_system) engine has become the norm. The reason this is called dynamic as oppose to static is that values on the page can change based on what is going on in the web app. In this article we will focus on a templating engine for the Express framework called [Pug](https://pugjs.org/api/getting-started.html).

**What Is Pug?**

In 2016 due to a copyright issue the templating engine known as [**jade**](https://jade-lang.com/) changed its name to [**pug**](https://github.com/pugjs/pug). Due to its default integration with Express.js it has become a popular option for rendering HTML templates within this framework. In order to install Pug as a dependency within an Express project, simply open a terminal within the root of your project and enter:

npm install --save pug

Once this finishes installing you should see an entry for it within your package.json file under the “dependencies” key. In order to start utilizing pug you need to make a few changes within the top most layer of your project (**app.js** in this case).

This includes adjusting a few keys within your app object that will change the behavior of express using [.set()](https://expressjs.com/en/5x/api.html#app.set). Namely you should set your ‘view engine’ key to ‘pug’ and optionally you can set ‘views’ to ‘views’. The later example however is the default behavior and can be omitted. If you are serving static HTML files you will also want to change your [res.sendFile](https://expressjs.com/en/api.html" \l "res.sendFile" \t "_blank) to [res.render](https://expressjs.com/en/api.html" \l "res.render" \t "_blank) within the middleware and name the pug template you wish to serve for that route. Once this is complete a very basic **app.js** might look something like this:

Text

Description automatically generated

If this looks a bit confusing a good breakdown can be found in my [previous blog post](https://ryan-m-schleck.medium.com/using-express-for-static-html-bc4235351473) regarding static html. You’ll notice on line 23 we are passing a variable **pageTitle** to our **home.pug** template.

You will also want to create a **home.pug** template within the **views** folder. Lets take a glance of what that might look like:

A screenshot of a computer

Description automatically generated with medium confidence

notice on line 7 we are utilizing our **pageTitle** variable we got from the locals object within our middleware (line 23 in app.js). I used comments in this picture to show what some of the html will look like above the Pug content which will display it.

Some of the initial differences you should notice is that there are no longer any closing tags and that attributes are passed to html elements via parentheses. I’m sure you also saw on line 7 that we used *#{ }* to render our local variables. One essential difference between this and traditional html that is NOT immediately apparent is that nesting in pug is done via indentation so spacing matters. Thus our h1 tag on line 14 is nested within our div what we start on line 12. Now that we have some of the basics lets delve into how we can utilize user submitted content with a separate example!

**Dynamic HTML Using Pug**

Lets go over how to incorporate dynamic html into an express app using pug! We will discuss this by using an example in which we have a form page (that creates pugs) and submitting this will let us see a list of pugs we have made. We are going to need to incorporate some routing in our entry point for our app (app.js) as well as some routes we will put into a route folder. Before we have a look at a particular file, the file structure for this should resemble:

Graphical user interface, text, application

Description automatically generated

You will notice that **app.js** is the top most layer of our app. Within it we are going to want to import the path module as well as express. We are also going to want to set up our parsing for requests in order to allow us to work with user submitted data. We are going to need at least 3 routes for our /pugs domain. This will allow us to create a pug via a form, deal with the logic of creating said pug with a post request, and read our list of created pugs.

We will move this logic into our routes folder within **pug.js** and import it within our top most layer for all routes that begin with ‘/pugs’. Once we finish all of this our **app.js** file should look something like this:

Text

Description automatically generated

As previously mentioned we will have 3 middleware functions within our **pug.js** file that resides within our routes folder. Within pug.js we are going want to import the path module as well as express again. We are also going to want to make use of the [express.Router](https://expressjs.com/en/guide/routing.html" \t "_blank) class in order to create a Router instance that will handle our various routes using methods resembling [http methods](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods) (sometimes called verbs).

Each one of these methods accepts a middleware function that express will execute whenever the request matches the route. Within 2 of these middleware functions we are going to want to call **[res.render](https://expressjs.com/en/api.html" \l "res.render" \t "_blank)** which will receive 2 arguments. The first of these arguments will be a string representing a file path to a .pug file within our views folder (due to lines 10 & 13 on **app.js**), and the second will be an object with keys that will be local variables available within our view file followed by their corresponding values. Lastly we will export our router instance containing all of our routes using [module.exports](https://nodejs.org/api/modules.html" \l "moduleexports" \t "_blank). The complete **pug.js** file should look something like this:

Text

Description automatically generated

With this all of our routing is done. We now need to work on our 2 views. You’ll notice line 28–30 in **app.js** mentions a 3rd view called *404* this is just an error page view that I don’t feel the need to go over.

I just included it because I feel it’s good practice to always include such a thing. The first of the 2 views we need will be the one mentioned in our first middleware function within **pug.js** (see picture above). It is the form a user will fill out to create a pug. The form will have names consistent with what we expect in our post middleware function (name, age, owner). Within our views folder we have a **pugs-new.pug** file which resembles this:

A screenshot of a computer

Description automatically generated with medium confidence

This form will submit to our ‘/pugs/create’ route via a POST request which will add our new pug object to our pugs array within **pug.js**and redirect us to our ‘/pugs’ route. It is important to note that this means any created pugs will be available for all requests across different clients. Meaning that this is not a great way to manage data, however it is simple and good for a demonstration on using express to display dynamic pages. Now that that disclaimer is out of the way lets move on to our dynamic html page. This will be a list containing all of our created pugs. In order to access our created pugs we iterate through our pugs array using an [each statement](https://pugjs.org/language/iteration.html). For each pug object we then create 3 variables through [object destructuring](https://www.javascripttutorial.net/es6/javascript-object-destructuring/) and display them using !{}. Despite numerous [posts on stack overflow](https://stackoverflow.com/questions/24316772/iterate-over-an-array-of-objects-in-jade-pugjs) and the [pug docs](https://pugjs.org/language/interpolation.html) suggesting you use different string interpolation for this. I’ve found that using li= or #{} within cases for which you are iterating over an array. And then using keys in an object will just cause pug to escape the html due to a formatting issue with the user data.

With this in mind our **pugs-index.pug** file should resemble something like this:

Text

Description automatically generated

**Conclusion**

With this our user should be able to submit data and have our app display it which meets our goal! For a better look at the templating engine called pug feel free to check out [their documentation](https://pugjs.org/api/getting-started.html). It’s important to note Pug is not the only player in the game as far as html templating options for Express.js. Checkout [this list](https://expressjs.com/en/resources/template-engines.html) of other options! Also if you are really curious about that 404 page I didn’t go into or want to fire up this app for yourself I’ve included a [repository on github](https://github.com/FluxOfPingIntegers/express-pug-demo) for this example. Thanks for reading!