Template engines Introduction

A template engine facilitates the use of static template files in your applications. At runtime, it replaces variables in a template file with actual values, and transforms the template into an HTML file sent to the client. So this approach is favoured because it allows us to design HTML pages easily.

Some popular template engines that work with Express are Pug, Mustache, and EJS. The Express application generator uses Pug (formerly known as jade) as its default one, but it also supports several others.
See Template Engines (Express wiki) for a list of template engines you can use with Express. See also Comparing JavaScript Templating Engines: Pug, Mustache, Dust and More.

Using template engines with Express

Template engine allows you to use static template files in your application. To render template files you have to set the following application setting properties:

- **Views:** It specifies a directory where the template files are located. **For example:** app.set('views', './views').
- **view engine:** It specifies the template engine that you use. For example, to use the Pug template engine: app.set('view engine', 'pug').

Let's take the Pug template engine.

Pug Template Engine

Let's learn how to use Pug template engine in Node.js applications using Express.js. Pug is a template engine for Node.js. Pug uses white spaces and indentation as part of its syntax. Its syntax is easy to learn.

Install Pug

Execute the following command to install Pug template engine:

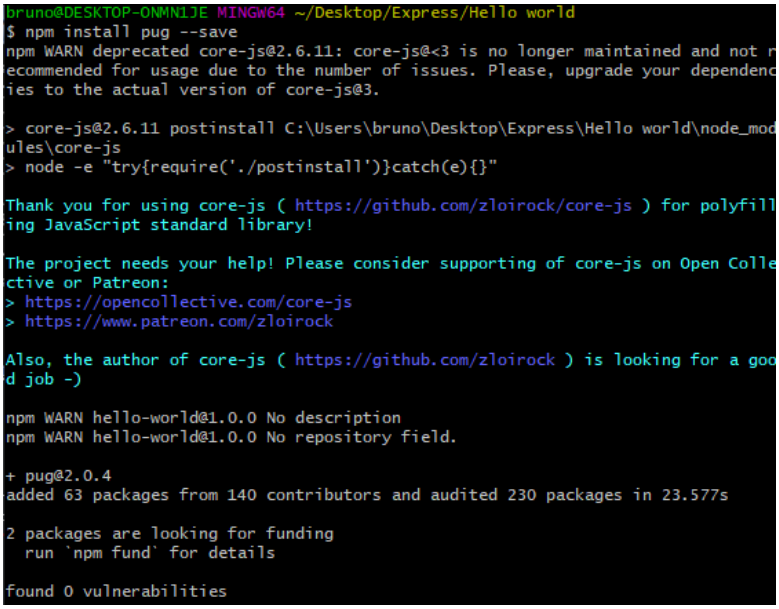
```
npm install pug --save
```

Pug Template Engine 1/2

Install pug

Execute the following command to install Pug template engine:

```
npm install pug --save
```



Pug Template Engine 2/2

Now that Pug is installed, set it as the default template engine for your app. You don't need to 'require' it. Add the following code to your **index.js** file.

```
app.set('view engine', 'pug');
app.set('views','./views');
```

Now create a new directory called "views". Inside it, create a file called first_view.pug, and enter the following data in.

```
doctype html
html
   head
      title = "Hello Pug"
   body
      p.greetings#people Hello World!
```

To run this page, add the following route to your app.

```
app.get('/first_template', function(req, res){
   res.render('first_view');
});
```

You will get the output: **Hello World!**

Pug converts this very simple looking markup into HTML. We don't need to close our tags nor use class and ID keywords. Instead, we use '.' and '#' to define them. The previous code first gets converted into:

```
<!DOCTYPE html>
<html>
   <head>
      <title>Hello Pug</title>
   </head>

   <body>
      <p class = "greetings" id = "people">Hello World!</p>
   </body>
</html>
```

Pug is capable of doing much more than simplifying HTML markup.

Important Pug Features (1/5)

Important Pug Features

Let us now explore a few important Pug features.

Simple Tags

Tags are nested according to their indentation. Like in the example before, `<title>` was indented within the `<head>` tag, so it was inside it. But, the `<body>` tag was on the same indentation and that means it is a sibling of the `<head>` tag.

We don't need to close tags. As soon as Pug encounters the next tag on the same or outer indentation level, it closes the tag automatically.

We have 3 methods for putting text inside a tag:

Space separated

```
h1 Welcome to Pug
```

Piped text

```
div
   | To insert multiple lines of text,
   | You can use the pipe operator
```

Block of text

```
div.
   But that gets tedious if you have a lot of text.
   You can use "." at the end of a tag to denote a block of text.
   To put tags inside this block, simply enter tag in a new line and
   indent it accordingly.
```

Comments

Pug uses the same syntax as JavaScript(//) for creating comments. These comments are converted into HTML comments(<!--comment-->). For example,

```
//This is a Pug comment

<!--This is a Pug comment-->
```

Attributes

To define attributes, we use a list of attributes that are separated by commas and put them inside parenthesis. Class and ID attributes have special representations. The following line of code covers defining attributes, classes and HTML tag ID.

```
div.container.column.main#division(width = "100", height = "100")
```

This line of code gets converted into the following:

```
<div class = "container column main" id = "division" width = "100" height = "100"></
```

Important Pug Features (2/5)

Passing Values to Templates

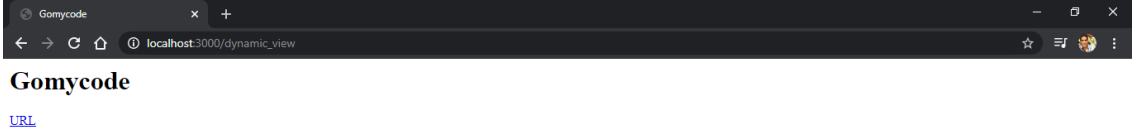
When we render a Pug template, we can actually pass it a value from our route handler, which we can then use in our template. Create a new route handler with the following.

```
const express = require('express');
const app = express();

app.get('/dynamic_view', function(req, res){
   res.render('dynamic', {
      name: “Gomycode",
      url:"http://www.tutorialspoint.com"
   });
});

app.listen(3000);
```

And create a new view file in views directory, called dynamic.pug, with the following code

```
html
   head
      title=name
   body
      h1=name
      a(href = url) URL
```

Open localhost:3000/dynamic_view in your browser; You should get the following output



Important Pug Features (3/5)

We can also use these passed variables within text. To insert passed variables in between text of a tag, we use #{variableName} syntax. For example, in the above example, if we wanted to put Greetings from TutorialsPoint, then we could have done the following.

```
html
   head
      title = name
   body
      h1 Greetings from #{name}
      a(href = url) URL
```

This method of using values is called interpolation. The above code will display the following output.

Important Pug Features. (4/5)

Conditionals

We can use conditional statements and looping constructs as well.

Consider the following

If a User is logged in, the page should display "Hi, User" and if not, then the "Login/Sign Up" link.

To achieve this, we can define a simple template like

```
html
   head
      title Simple template
   body
      if(user)
         h1 Hi, #{user.name}
      else
         a(href = "/sign_up") Sign Up
```

When we render this using our routes, we can pass an object as in the following program

```
res.render('/dynamic',{
   user: {name: "Ayush", age: "20"}
});
```

You will receive a message – Hi, Ayush. But if we don't pass any object or pass one with no user key, then we will get a signup link.

Important Pug Features (5/5)

Include and Components

Pug provides a very intuitive way to create components for a web page. For example, if you see a news website, the header with logo and categories is always fixed. Instead of copying that to every view we create, we can use the include feature. Following example shows how we can use this feature.

Create 3 views with the following code

HEADER.PUG

```
div.header.
   I'm the header for this website.
```

CONTENT.PUG

```
html
   head
      title Simple template
   body
      include ./header.pug
      h3 I'm the main content
      include ./footer.pug
```

FOOTER.PUG

```
div.footer.
   I'm the footer for this website.
```

Create a route for this as follows

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

app.get('/components', function(req, res){
    res.render('content');
});

app.listen(3000);
```

Go to localhost:3000/components, you will receive the following output

include can also be used to include plaintext, css and JavaScript.

There are many more features of Pug. But those are out of the scope for this tutorial. You can further explore Pug at Pug

Previous next