Sistemas de templates

¿Qué son?

Los sistemas de templates se basan en tener un objeto como vista y una parte de template que por identificadores imprimirán partes de la vista en donde nosotros deseemos, por ejemplo, los que ya hayan leído la lectura de mustache.js verán que tenemos una vista y en otra variable un template en donde colocamos variables que identifican la información que deseamos imprimir, y al momento de renderizar nos aparecerá automáticamente toda nuestra información de una manera mucho más escalable.

Para que me sirven los templates del lado del cliente?

Actualmente consumimos muchas API's desde JavaScript basadas en JSON en donde nos toca recorrer cada objeto y empezar a crear cadenas de html para luego acceder al dom y tener que imprimirlos.

Esto resulta bien cuando nuestra app es muy pequeña y somos adolescentes en el desarrollo del Frontend, pero como todos sabemos nuestras apps crecerán y esta técnica se vuelve tediosa y poco escalable y es aquí en donde los muy profesionales templates llegan a salvarnos convirtiéndose mucho más escalables y menos repetitivos.

Algunos sistemas de templates:

Swig

Lo vemos como principal sistema de templates usado por Django y también aplicado en Node.js y JavaScript, de forma parecida a mustache vamos a manejar doble llaves para colocar las variables y {% %} para condicionales y bucles.

Podemos usarlo de la siguiente forma:
Instalamos Swig en node.js de forma global de la misma forma que lo hacemos con stylus

```
npm install swig
```

Creamos nuestro template

```
<h1>{{ pagename|title }}</h1>
<ul>
{% for author in authors %}
  <li{% if loop.first%} class="first"{% endif %}>
    {{ author }}
  </li>
{% else %}
  <li>There are no authors.</li>
{% endfor %}
</ul>
```

Renderiamos el template

```
var template  = require('swig');
var tmpl = template.compileFile('/path/to/template.html');
tmpl.render({
    pagename: 'awesome people',
    authors: ['Paul', 'Jim', 'Jane']
});
Resultado Final
<h1>Awesome People</h1>
<ul>
  <li class="first">Paul</li>
  <li>Jim</li>
  <li>Jane</li>
</ul>
```

Jade

Para los que manejan Ruby On Rails se les hará muy familiar y se basa en una filosofía parecida a la de stylus pero con html5 en donde la

indentación y el uso de etiquetas sin los símbolos < o > las identifica automáticamente, lo usamos de la siguiente forma:

Documento en jade

```
doctype 5
html(lang="en")
  head
    title= pageTitle
    script(type='text/javascript')
      if (foo) {
         bar()
      }
  body
    h1 Jade - node template engine
    #container.col
      if youAreUsingJade
        p You are amazing
      else
        p Get on it!
      p.
        Jade is a terse and simple
        templating language with a
        strong focus on performance
        and powerful features.
```

Documento rendereado

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Jade</title>
    <script type="text/javascript">
      if (foo) {
        bar()
      }
    </script>
  </head>
  <body>
    <h1>Jade - node template engine</h1>
    <div id="container" class="col">
      <p>You are amazing</p>
      <p>Jade is a terse and simple
         templating language with a
         strong focus on performance
         and powerful features.</p>
    </div>
```

```
  </body>
</html>
```

Mustache.js

En este no sere muy extenso, trabaja de una forma muy parecida a la de swig en base de llaves, manejamos una vista en objetos y puedes aprender a usarla de una forma mas profesional en la lectura disponible de la clase 3.

Underscore.js

Es una libreria que abarca muchas mas cosas que un template y lo usamos casi siempre que trabajamos con backbone.js al ser parte de estas:

```
$(document).ready(function () {
  var template = _.template("<p>Hola <%= nombre %></p>");

  // Ejecutamos la función regresada y le pasamos como parámetro
  // las variables y luego con jQuery la metemos al div con id contenido
  $('#contenido').html(template({nombre: "Isaac"}));
});
```

_.template() nos definirá el template con el que se renderiza y el identificador de las variables será <%= .. %> y para que renderize colocamos la penúltima línea en donde pasamos la vista con la que se completara el template.