**Mustache**

Mustache — шаблонизатор, который содержит минимум управляющей логики и доступен для разных языков программирования. Его можно использовать и на сервере (PHP, Ruby и т.д.), и на клиенте (Javascript).

Если вы динамически подгружали с сервера блоки HTML-кода, а не структурированные данные только потому, что не хотели дублировать рендеринг на сервере и на клиенте, то Mustache поможет этого избежать.

Шаблон пишется на простом языке, состоящем из нескольких типов тегов. Теги обрамляются двумя или тремя фигурными скобками с каждой стороны. Можно использовать вложенные шаблоны.

Простой пример шаблона:

<h1>{{header}}</h1>

<p>{{content}}</p>

<ul>

{{#authors}}

<li>{{#accent}} {{.}} {{/accent}}</li>

{{/authors}}

{{^authors}}

<li>anonymous</li>

{{/authors}}

</ul>

Данные, с которыми работает шаблон, называются контекстом. Имя тега указывает, к какому полю контекста необходимо обратиться. Пример данных, которые могут послужить контекстом для нашего шаблона:

var data = {

header : "Новый пост",

content: "Первая строка<br />Вторая",

authors: ["alex", "daemon", "john"],

accent : function () {

return function (text, render) {

text = render(text);

return '<strong>' + text + '</strong>';

}

}

};

Чтобы «запустить» шаблонизатор и отрисовать с помощью шаблона данные, необходимо подключить библиотеку и вызывать рендеринг методом Mustache.to\_html(template, data);

Всего в Mustache четыре основных типа тегов: переменная, секция, комментарий и подключение дополнительного шаблона.

*Переменная* выводит данные с экранированием HTML-сущностей {{header}} и без экранирования {{{content}}}. Отличаются они количеством скобок. В нашем случае, вместо {{header}} подставится строчка «Новый пост».

*Секция* представляет собой парный тег. Принцип ее действия зависит от типа данных, с которыми она работает. Если в контексте имени секции соответствует поле со списком, то шаблонизатор проходит по его элементам и текст внутри парного тега обрабатывается по одному разу для каждого элемента списка. Элемент списка подставляется заместо тега-точки. Так, например, секция {{#authors}}<li>{{.}}</li>{{/authors}} превратится в <li>alex</li><li>daemon</li><li>john</li>. Если список пуст, то обрабатывается содержимое «тега с крышечкой», в нашем случае — это {{^authors}} … {{/authors}}.

Если имени секции соответствует функция, то для подстановки будет использован результат ее выполнения.

Если поле, соответствующее имени секции, не является ни списком, ни функцией, то оно будет использоваться в качестве контекста для обработки содержимого тега.

*Комментарий* оформляется в виде тега с восклицательным знаком, например, {{! comment content}}.

*Подключение* дополнительного шаблона вызывается с помощью тега с угловой скобкой. Например, {{>copyright}}. Если в текущем контексте присутствует поле с таким названием, то оно будет передано в качестве контекста для подключаемого шаблона.

**Handlebars**

Первое что нужно сделать — подключить Handlebars

<script src="js/handlebars.js"></script>

Handlebars генерирует HTML из JSON данных. Для того, чтобы браузер распознал шаблон Handlebars, мы должны присвоить ему тип text/x-handlebars-template, а также ID, чтобы ссылаться на него позже. Разметка для создания шаблона, который будет получать имя и выводить HTML может выглядеть следующим образом:

<ul class="updates">

<script id="template" type="text/x-handlebars-template">

<li>

<h2>{{name}}</h2>

</li>

</script>

</ul>

Имя будет получено из JSON и подставлено вместо {{name}}. Мы создали простой JSON объект, содержащий одно свойство name со значением 'John Doe', которое мы хотим поместить в HTML. Но прежде нам нужно скомпилировать код шаблона и присоединить его к классу .updates в HTML, передав JSON объект data как параметр:

var template = Handlebars.compile( $('#template').html() );

$('.updates').append( template(data) );