No Templates

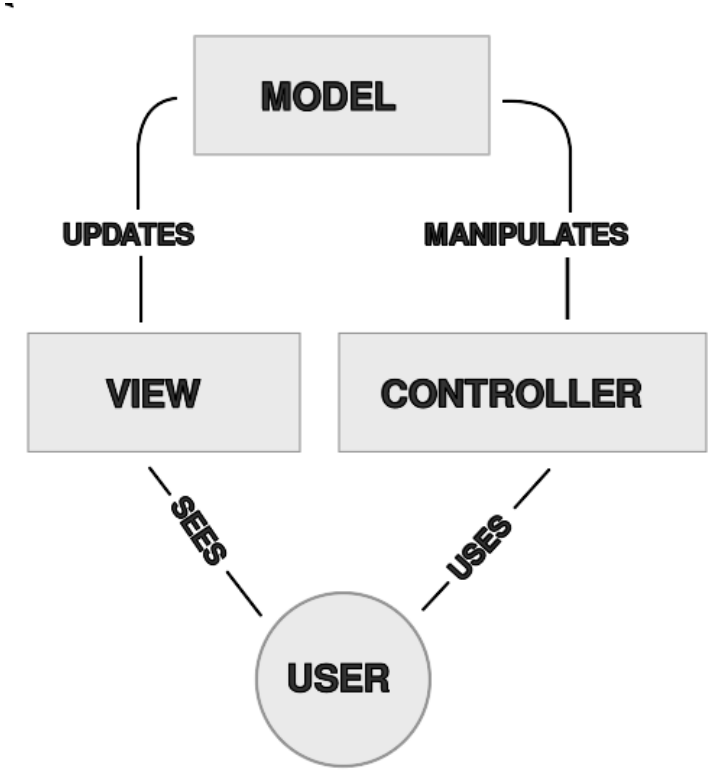
- Produce HTML within code.
- HTML strings interspersed with code.
- No separation of content and presentation.
- Unmaintainable.

Model View Controller

- MVC provides clear separation between model, view and controller.
- The **model** represents the domain knowledge, AKA business logic.
- User interacts with a **controller** which manipulates the model.
- When the model changes, it updates one or more **views** which are seen by the user.

Model View Controller Continued

From wikipedia:



Templates

- Templates are well suited to build views for document oriented view technologies like HTML.
- A **template processor** or **template engine** mixes a model into a template to produce a view.
- Typical template consists of multiple languages:
  1. View technology language like HTML.
  2. Template language.

Template Requirements

- Some way of distinguishing fragments in template language from view language (HTML).
- Template language **must** have some way of inserting model values into view.
- Template language should allow conditional generation of a fragment based on model conditions.
- Template language should allow repetitive generation of a fragment based on model parameters.

Template Anti-Requirements

- Should not allow general programming logic.
- Definitely should not allow database access.
- *Java Server Pages*, Microsoft's *Active Server Pages* get it wrong:
  - Tied to a particular model (web).
  - Requires discipline on part of the user to not mix model logic into view.

Mustache Overview

- Logic-less templates.
- Independent of target view technology; can be used to generate HTML, CSS, XML, configuration files, etc.
- Can be used from a wide variety of languages: JavaScript, Java, C#, Python, Ruby, C++, Swift among others.
- Uses {{ and }} delimiters.
- Mustache uses the term "view" to refer to the *view-model* which is rendered into what MVC calls a "view".

Mustache Example

In simple.js. Given template:

```
Hello {{name}}, you are {{age}} years old
```

and model for view:

```
{ name: 'John', age: 10 }
```

will render as:

```
Hello John, you are 10 years old
```

Mustache HTML Escaping

`{{...}}` escape characters special to HTML; to avoid escaping behavior, use `{{{...}}}`.
html-esc.js outputs:

```
Expression a &lt; 2 means <var>a</var> <em>less-than</em> 2
```

Mustache Conditional Rendering of Sections

A section is delimited by `{{#...}}` and `{{/...}}` delimiters. Can be used for conditionals and iteration.
If in model, section key is a truthy non-list value, then section will be rendered.
cond.js outputs:

```
Hello John  you are 10 years old
```

Mustache Inverted Sections

If section key is preceeded by a ^ instead of a #, the section is rendered only when the identifer is falsy in the model.
not.js outputs:

```
Hello John,  sorry, I don't know your age
```

Mustache Rendering Lists

If in model, the value of the section key is a list, then it will render section for each element of the list with the context set to the list element.
list.js outputs:

```
Hello John, you are 10 years old
Hello Mary, you are 15 years old
```

Mustache Context Search

If key is not found, recursively looks for key in surrounding context: lookup.js outputs:

```
Hello John, you are 10 years old
Goodbye Mary, you are 15 years old
```

Mustache Functions

If value of section key is a function, then the function is called with 2 arguments:

1. The text of the section body.
2. A special `render` function which uses the current view as its view context.

`this` is set to current view context.
fn.js outputs:

```
Hello <b>John</b>, you are 10 years old
```

Complex Example

Example uses function to build html class dynamically and outputs

```
<h1>Report</h1>
<table>
  <tr>
    <th>Question></th>
    <th>a</th>
    <th>b</th>
    <th>c</th>
  </tr>
   <tr>
     <td>1</td>
     <td>10%</td>
     <td>20%</td>
     <td class="correct">70%</td>
   </tr>
```

Complex Example Output Continued

```
    <tr>
      <td>2</td>
      <td>10%</td>
      <td class="correct">40%</td>
      <td class="distract">50%</td>
    </tr>
</table>
```

Mustache Partial

Optional third argument to `Mustache.render()`: object giving mapping for secondary template names.

- Use {{> *name*}} to **include** secondary template.
- Rendered using current context.

partial.js outputs:

```
Hello John, you are 10 years old
```

Recursive Partials

- Partial can be invoked recursively to output nested structure with nesting structure specified only by data.
- Imperative that each node specify list of child nodes (even if empty) to avoid recursive loop.

Subsequent slide shows output of recursive.js after whitespace cleanup

Recursive Partials Output

```
<span class="name">System</span>
<ul>
  <li><span class="name">Display</span>
    <ul>
      <li><span class="name">Monitor 1</span></li>
      <li><span class="name">Monitor 2</span></li>
    </ul>
  </li>
  <li><span class="name">Processor</span>
    <ul>
      <li><span class="name">CPU</span></li>
      <li><span class="name">Memory</span></li>
    </ul>
  </li>
  <li><span class="name">Keyboard</span></li>
  <li><span class="name">Mouse</span></li>
```

References

Mustache Language Manual
Mustache Processor Manual
Christopher Coenraets, *Tutorial: HTML Templates with Mustache.js*.