Templating

How Templating Works

- A "template language" has a defined syntax beyond plain HTML
- That language and syntax creates dynamic HTML based on provided data
- The templating engine compiles the data and template down to plain HTML for the browser

Most Common Templates

- Embedded JavaScript (EJS) - Not as common as the other two, but worth mentioning. Parallels how ERB (Embedded RuBy) works in Ruby.
- Pug (Formerly Jade) - The default for Express applications. Parallels how HAML (HTML Abstration Markup Language) works in Ruby.
- Handlebars - Another popular templating solution.

Setting Up Template Rendering in Express

- Set up the application's `view engine` valuee
- Optionally set up the root directory for the views in the application's `views` value. Defaults to the `views` directory at the root of the project.

```
var express = require('express');
var app = express();
//set the view engine to hbs for handlebars, ejs for ejs, or pug for pug
app.set('view engine', 'ejs');
//sets this application to look at `my-views` next to the running application
app.set('views', './my-views');
```

- You will also need to install the templating engine you wish to use, eg `npm install --save pug`.

Other Templating Engines

- Express supports many, many more templating libraries. Naming a few:
  - `eco` - embedded CoffeeScript
  - `hogan` - JavaScript templating from Twitter
  - `nunjucks` - templating supported by Mozilla

Rendering With Templates

- Express' `response` object exposes `.render()`
- This allows us to render a template for a path on disk, automatically responding with compiled HTML from the template and supplied data

```
app.get('/home', function(req, res){
    //renders the `home-page` view in `views`
    res.render('home-page');
});
```

Rendering a Template with Data

- The `.render()` method allows for passing data to the template for rendering and logic

```
app.get('/home-with-data', function(req, res){
    res.render('grocery-list', {
        groceries:[
        'bananas',
        'milk',
        'lettuce
    ]
    })
})
```

Embedded JavaScript (EJS)

- Source in <%   %> is executed (meaning you can leverage regular JavaScript constructs like loops), <%=   %> adds HTML to the result.
- HTML is otherwise normal
- Can be used directly in the browser

```
<div>
    <h1>My Grocery List</h1>
    <ul>
    <% for(var i = 0; i < groceries.length; i++) { %>
    <li><%= groceries[i] %></li>
    <% } %>
    </ul>
</div>
```

Exercise: EJS

- Create a new project directory called `ejs-exercise` and `cd` into it.
- Run `npm init`, answering the questions
- Install and save `express` and `ejs`
- Create a `views` directory
- Create a view called `list.ejs` in `views` to display a modified grocery list: display each store with its respective list
- Create an Express server to render the template on `/`

Pug (formerly Jade)

- Very condensed version of HTML that is whitespace-sensitive
- *Not* built for browser use

```
div
    h1 My Grocery List
    ul
    each val in groceries
        li= val
```

Exercise: Pug

- Create a new project directory called `pug-exercise` and `cd` into it.
- Run `npm  init`, answering the questions
- Install and save `express` and `pug`
- Create a `views` directory
- Create a view in `views` called `list.pug` to display a modified grocery list: display each store with its respective list
- Create an Express server to render the template on `/`

Handlebars

- Handlebars handles constructs through pluggable helpers (prefixed with #)
- Uses `this` for current items instead of an array index or temporary variable
- Can be used directly in the browser

```
<div>
    <h1>My Grocery List</h1>
    <ul>
    {{#each groceries}}
    <li>{{this}}</li>
    {{/each}}
    </ul>
</div>
```

Exercise: Handlebars

- Create a new project directory called `hbs-exercise` and `cd` into it.
- Run `npm  init`, answering the questions
- Install and save `express` and `hbs`
- Create a `views` directory
- Create a view in `views` called `list.hbs` to display a modified grocery list: display each store with its respective list
- Create an Express server to render the template on `/`

EJS: Partials

- Use `<% include template-name.ejs %>` to pull in partial templates.
- EJS renders the partial as though it were a part of the template it was included from all along, meaning variables are implicitly available within the included partials

  ```html
  <!-- item.ejs -->
  <li><%= groceries[i] %></li>
  ```

```
<!-- groceries.ejs -->
```

Pug: Partials

- Use `include template-name.pug` to pull in partial templates.
- Pug also renders the partial as though it were a part of the template it was included from. Variables are inherited from the parent template and are implcitly available.

```
html
//item.pug
li= val

html
```

Handlebars: Partials

- Handlebars is a little different in that the partials have to be read and registered first

```js
var express = require('express');
var hbs = require('hbs')
var fs = require('fs');
var app = express();

//include the item partial
hbs.registerPartial('item', fs.readFileSync('./views/item.hbs', 'utf-8'));
app.set('view engine', 'hbs');
```

Exercise: Partials

- Modify each of your templates from before to break out the store into a `store` partial, with each item in an `item` partial.