Express #2:
View Engines & Templates

Rendering As We Know It

- So far, we've rendered static HTML files that we then make dynamic on the client using Javascript
- This has worked well because most of our dynamic content has come from user input (song files, todo list items etc.)
- But now that we're dealing with data in Node, we need that dynamic content to work on our server, not the client

Example Problem: Menu Website

- We're making a website for a client who runs a cafe
- They provide us an array of objects for menu items:
- How do we render this menu in HTML?

```
const menu = [{
    item: "Coffee",
    price: 2.95,
}, {
    item: "Croissant",
    price: 3.95,
}, /* rest of the menu... */]
```

Example Solution: Doing it Manually

- We could just write out each element manually in HTML:
- But this is time consuming, and won't update with our array
- Plus, we're programmers! We *hate* doing things by hand...

```
<ul class="menu">
    <li class="menu-item">Coffee: 2.95</li>
    <li class="menu-item">Croissant: 3.95</li>
    <!-- Rest of the menu... -->
</ul>
```

Example Solution: Add them via jQuery

- We could keep the menu in JS, and add each item in jQuery
- But then the user might briefly see an empty menu list
- And search engines or users without JS won't see it at all

```
var $menu = $(".menu");

for (var i = 0; i < menu.length; i++) {
    var item = menu[i].item + ": $" = menu[i].price;
    $menu.append('<li class="menu-item">' + item + "</li>");
}
```

Example Solution: Use a View Engine

- We briefly encountered view engines in Express earlier, but didn't use it other than to output HTML we hard-coded.
- However, most also allow for dynamic content, not just static HTML.
- If we could iterate over the `menu` array in our template, we could serve up the right HTML.
- For example, it could looks something like...

```
<ul class="menu">
    <!-- For every menu item... -->
        <li class="menu-item">
            {item}: ${price}
        </li>
    <!-- End for -->
</ul>
```

View Engines: EJS

- EJS -> "Embedded JavaScript", because we *embed* JS in our template with it
- We can use <% to enter JS mode, and %> to leave it
- We can also use <%= or <%– to print out escaped or unescaped values
- With these combined, we can print variables, elements in a loop, or conditionally

```
# In terminal
npm install --save ejs

// In app.js
app.set("view engine", "ejs");
```

EJS Example: Printing variables

```
<div class="user">
    <img class="user-photo" src="<%= user.photo %>"/>
    <div class="user-name">
        <%= user.name %>
    </div>
    <div class="user-location">
        <%= user.city %>, <%= user.state %>
    </div>
</div>
```

EJS Example: Conditionals

```
<div class="nav">
    <a class="nav-logo" href="/">Home</a>
    <% if (isLoggedIn) { %>
        <a class="nav-link" href="/logout">Logout</a>
    <% } else { %>
        <a class="nav-link" href="/login">Login</a>
    <% } %>
</div>
```

View Engines: Pug

- Pug (formerly known as Jade) is an engine that aims to *replace* HTML, rather than simply enhance it like EJS
- Pug does not accept regular HTML, but instead a "cleaner" stripped down version that only takes in the bare minimum
- Rather than explaining, let me just show you some examples:

```
# In terminal
npm install --save pug
```

```
// In app.js
app.set("view engine", "pug");
```

Pug Example: Before (HTML)

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>Pug</title>
    </head>
    <body>
        <h1>Pug - node template engine</h1>
        <div id="container" class="col">
            <p>You are amazing</p>
            <p>
                Pug is a terse and simple templating language with
                a strong focus on performance and powerful features.
            </p>
        </div>
    </body>
</html>
```

Pug Example: After (Pug)

```
doctype html
html(lang="en")
    head
        title= pageTitle
    body
        h1 Pug - node template engine
        #container.col
            if youAreUsingPug
                p You are amazing
            else
                p Get on it!
            p.
                Pug is a terse and simple templating language with a
                strong focus on performance and powerful features.
```

How Do We Use Them?

- Once you've installed the module, and `app.set` the view engine, that's it
- However, the real power only comes out if the templates have variables
- We pass these variables as a second object argument to `res.render`
- The keys on this object are available as variables in the template:

```
// Inside a route in app.js
res.render("template", {
    title: "My website",
    message: "Hello!"
});
```

```
<!-- views/template.ejs -->
<html>
    <head>
        <title><%= title %></title>
    </head>
    <body>
        <h1><%= message %></h1>
    </body>
</html>
```

Exercise: Replace Variables

- Choose EJS or Pug as your view engine
- Take one of your portfolio site pages, and find anything that could be a variable (image URL, your title, a list of skills etc.)
- Replace the HTML with variables in your respective view engine
- Add the variables you replaced to your res.render call
- *Bonus Challenge*: Move the variables in to a JSON file, and `require` that file instead of hard-coding the values in app.js

Templates

- At this point, we've made a few HTML pages, and it's getting old
- Most everything in the head tag is identical across your pages
- If you have a header or a footer, you copy and paste them
- Need to make a change to that header or footer? Ha, you're making that change *to each and every page's file*.
- It would be nice if we could have one file that every page referenced for these common elements...

Includes to the Rescue

- In addition to loops, conditions, and printing values, most templating languages also support file includes
- What this means is that one template file can include another, and pass it variables as well
- Using this, we can make one master template file that includes individual pages

Example: Template & Include

- We define one template file that contains all of the common elements

```
<!-- views/template.ejs -->
<html>
    <head>
        <title>Will's Cool Site - <%= title %></title>
        <link rel="stylesheet" type="text/css" href="/css/style.css"/>
    </head>
    <body>
        <h1>Will's Cool Site</h1>

        <main>
            <%- include("pages/" + page, {
                any: "args",
                you: "want",
                in: "pages/{page}.ejs",
            }); %>
        </main>

        <footer>© Will O'Beirne 2017</footer>
    </body>
</html>
```

Example: Template & Include (cont.)

- Then we can re-use it by rendering the template, and passing it the relevant arguments

```
// app.js
app.get("/", function(req, res) {
    res.render("template", {
        page: "homepage",
        title: "Home",
    });
});

app.get("/about", function(req, res) {
    res.render("template", {
        page: "about",
        title: "About",
    });
});
```

Example: Template & Include (cont.)

```
<!-- views/pages/homepage.ejs -->
<p>
    Welcome to my homepage! It
    sure is great, ain't it.
</p>

<!-- views/pages/about.ejs -->
<p>
    This page was made by Will
    O'Beirne. He's pretty cool.
</p>
```

Exercise: Templatize Your Portfolio Site

- Now that we have the ability to use templates, let's make one for your portfolio site
- Determine what parts of your site should stay the same across all pages (Header, Footer, CSS, etc.) and what parts should be different
- Make a template file for all the parts that are the same
- Move the different parts into their own page files
- Change your routes to res.render the template, and have the template include the pages

Additional Resources

- Ejs.co - Official EJS site with examples and docs
- Pugjs.org - Official PUG site with examples and docs
- Express' Guide to Templating Engines