Express – {Templating, Forms}

CS 390 – Web Application Development

J. Setpal

October 16, 2023

Outline

Outline

WIWYT – Template Engines (Recap)

- Speeds up writing HTML by building off a template on the fly.
- Enables us to serve dynamic content using server-side rendering.

WIWYT – Form Handling

- Forms are great because they let us information in-take.

WIWYT – Form Handling

- Forms are great because they let us information in-take.
- BUT – they're exhausting to work with.

WIWYT – Form Handling

- Forms are great because they let us information in-take.
- BUT – they're exhausting to work with.
- Express can do the heavy lifting!

Outline

What's a Template Engine? (Recap)

Template Engines are used to ease and automate writing HTML.

There are several popular engines - today we'll be looking at **pug**: `https://pugjs.org/api/getting-started.html`.

It uses a markdown-like syntax. Has features like conditions, loops, includes & mixins.

Pug Syntax (Recap)

Each element is **only defined once**. Indentation specifies scope.

a. Elements are `div` by default.
b. `#<var>` after the element specifies the element id.
c. `.<var>` after the element specifies the element class(es).
d. Elements can be made multiline using a '`.`' at the end of the element.
e. Javascript can be injected using '`-`' at the beginning of the line.
f. Attributes can be specified using `elem(key="val" key2="val")`.

These are each included in a `.pug` file. This generates HTML output that can be rendered on-the-fly or statically.

Loops

Pug can loop through an array or object to procedurally render elements.

Loops

Pug can loop through an array or object to procedurally render elements.

Syntax:

```
for/each <var> in <array/object>
        <elem>= <var>
        ... additional interesting code
```

Example:

```
ul
   for i in [0, 1, 2, 3]
        li= i
```

Loops

Pug can loop through an array or object to procedurally render elements.

Syntax:

```
for/each <var> in <array/object>
        <elem>= <var>
        ... additional interesting code
```

Example:

```
ul
   for i in [0, 1, 2, 3]
        li= i
```

The input array can be specified dynamically by supplying a variable through express.

Loops

Pug can loop through an array or object to procedurally render elements.

Syntax:

```
for/each <var> in <array/object>
        <elem>= <var>
        ... additional interesting code
```

Example:

```
ul
   for i in [0, 1, 2, 3]
        li= i
```

The input array can be specified dynamically by supplying a variable through express.

`else` can be used to specify default behavior when no items are present to iterate through.

Conditionals #1

Pug implements if/else and switch statements to conditionally render elements.

Conditionals #1

Pug implements if/else and switch statements to conditionally render elements.

Syntax:

```
if <condition>
        ... stuff to render
else if <condition>
        ... stuff to render
else <condition>
        ... stuff to render
```

Example:

```
- const book = {genre: "horror", fiction: true}
if book.fiction
        p= book.genre
```

Conditionals #2

Switch is helpful when evaluating categorical values.

Conditionals #2

Switch is helpful when evaluating categorical values.

Syntax:

```
case <var>
        when <value>
                ... stuff to render
        when <value>
                ... stuff to render
```

Example:

```
- const book = {genre: "horror", fiction: true, rating: 10}
case book.genre
        when "horror"
                p= book.rating
        when "sci-fi"
                strong 10/10 best book ever
```

Includes

Pug, fundamentally, is build around the idea of *minimizing* how much we type.

Therefore, it integrates **includes** and **mixins** to follow DRY.

Includes

Pug, fundamentally, is build around the idea of *minimizing* how much we type.

Therefore, it integrates **includes** and **mixins** to follow DRY.

Includes are static renderable chunks of templates, that can be re-used in various template files.

Includes

Pug, fundamentally, is build around the idea of *minimizing* how much we type.

Therefore, it integrates **includes** and **mixins** to follow DRY.

Includes are static renderable chunks of templates, that can be re-used in various template files.

They are added using `include /path/to/file.pug`.

Mixins

Mixins are cross between *functions* and *includes*.

Mixins

Mixins are cross between *functions* and *includes*.

You can specify the location at which a chunk is rendered, similar to includes. However; unlike includes, mixins are **not restricted to static data**.

Mixins

Mixins are cross between *functions* and *includes*.

You can specify the location at which a chunk is rendered, similar to includes. However; unlike includes, mixins are **not restricted to static data**.

Syntax:

```
mixin func(var1, var2)
        p= var1 + ' and ' + var2

+func('Hello', 'World')
```

Express Integration

To integrate pug with express, you can do the following:

a. Create your views in a folder.

Express Integration

To integrate pug with express, you can do the following:

a. Create your views in a folder.

b. Set this folder as '`views`' using `app.set`.

Express Integration

To integrate pug with express, you can do the following:

a. Create your views in a folder.
b. Set this folder as '`views`' using `app.set`.
c. Set '`view engine`' as '`pug`' using `app.set`.

Express Integration

To integrate pug with express, you can do the following:

a. Create your views in a folder.
b. Set this folder as '`views`' using `app.set`.
c. Set '`view engine`' as '`pug`' using `app.set`.
d. In the route response, specify `res.render(<file>, {<key>: <value>});` to dynamically render the file server-side.

Let's Build a File Host Frontend!

If you can view this screen, I am making a mistake (again).

Outline

Forms – GET v POST

There are two primary ways for forms to send data to a monitoring route.

Forms – GET v POST

There are two primary ways for forms to send data to a monitoring route.

a. **GET:**

- Integrates data within the URL.

Forms – GET v POST

There are two primary ways for forms to send data to a monitoring route.

a. **GET:**

- Integrates data within the URL.
- Cached by default.

Forms – GET v POST

There are two primary ways for forms to send data to a monitoring route.

a. **GET:**

- Integrates data within the URL.
- Cached by default.
- Cannot handle sensitive data.

Forms – GET v POST

There are two primary ways for forms to send data to a monitoring route.

a. **GET:**
   - Integrates data within the URL.
   - Cached by default.
   - Cannot handle sensitive data.

b. **POST:**
   - Integrates data within the request body.

Forms – GET v POST

There are two primary ways for forms to send data to a monitoring route.

a. **GET:**
   - Integrates data within the URL.
   - Cached by default.
   - Cannot handle sensitive data.

b. **POST:**
   - Integrates data within the request body.
   - Doesn't cache data by default.

Forms – GET v POST

There are two primary ways for forms to send data to a monitoring route.

a. **GET:**
   - Integrates data within the URL.
   - Cached by default.
   - Cannot handle sensitive data.

b. **POST:**
   - Integrates data within the request body.
   - Doesn't cache data by default.
   - Ideal for sensitive information.

Forms – Workflow

Here's how we build a form using Express:

a. Have a `.pug` file that contains a form.

Forms – Workflow

Here's how we build a form using Express:

a. Have a `.pug` file that contains a form.
b. Setup a route that can host the form.

Forms – Workflow

Here's how we build a form using Express:

a. Have a `.pug` file that contains a form.
b. Setup a route that can host the form.
c. Direct the response to a route monitored by Express.

Forms – Workflow

Here's how we build a form using Express:

a. Have a `.pug` file that contains a form.
b. Setup a route that can host the form.
c. Direct the response to a route monitored by Express.
d. Handle the data!

Let's Build a Basic Form!

If you can view this screen, I am making a mistake (again again).

Outline

Homework 3

Change of Plan: Homework 3 covering part 1 of Node was supposed to be released last Wednesday, but we were only able to complete Templating today.

So, Homework 3 will be a larger assignment with 2x the time and point value, and covers concepts throughout the Node / Express.js module.

Thank you!

Have an awesome rest of your day!

Slides: `https://www.cs.purdue.edu/homes/jsetpal/slides/templating,forms.pdf`

If anything's incorrect or unclear, please ping: `jsetpal@purdue.edu`
I'll patch it ASAP.