HTML Forms

Lecture 8
Chapter 6 in HTML text

Back to HTML

We've now learned a lot about JavaScript, and are ready to write some server-side code to handle user input.

So how do we get user input from the browser to the server? HTML Forms!

First, some Node stuff

- We're going to have to get better at handling HTTP. We could enhance our own wserver.js - but Node has many packages to help us.
- **npm** is the Node Package Manager
- A popular middleware package is called connect.
- From the command line, type

```
npm install connect
```

  - Note - you should be in the directory where your JavaScript code will be.

connect package

connect lets us build a robust HTTP server with only a few lines of code.

It includes static resource (html file) serving, logging, and dynamic templated content

```
var connect = require("connect");
var http = require("http");
var app = connect()
  .use(connect.logger('dev'))
  .use(connect.static('public'))
  .use(function(req, res){
    res.end('hello world\n');
  });
 http.createServer(app).listen(3000);
```

We'll discuss **require**, in more depth when we turn our attention to the details of Node.js

Understanding the server

```
var connect = require("connect");
var http = require("http");
var app = connect()
  .use(connect.logger('dev'))
  .use(connect.static('public'))
  .use(function(req, res){
    res.end('hello world\n');
  });
 http.createServer(app).listen(3000);
```

Configures logging and the use of the **public** directory to serve static content

An anonymous function which will be called last. The response will be closed already if static content matching the URL was found in *public* directory

req is an **object** containing the HTTP request data
res is an object representing the response object (and stream)
The **end** function sends the string, and closes the stream (HTTP status = 200)

The request object

Lets inspect the request object a bit by adding to the function (and naming it)

```
var app = connect()
  .use(connect.logger('dev'))
  .use(connect.static('public'))
  .use(serve)
http.createServer(app).listen(3000);
function serve (req, res) {
    console.log("Host name:  " + req.headers.host);
    console.log("Connection:  " + req.headers.connection);
    console.log("Accept:  " + req.headers.accept);
    res.end('hello world\n');
}
```

HTML Forms

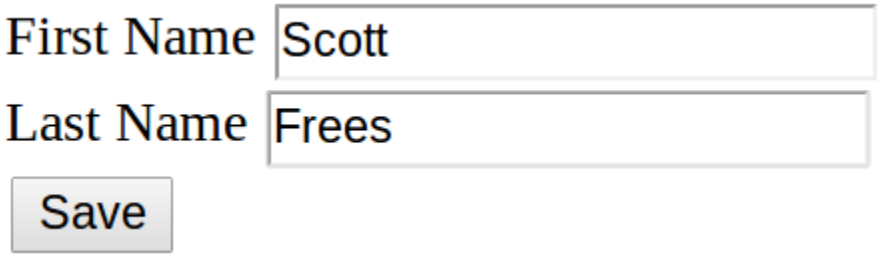
OK - so lets see how we can pass more data over the wire.

A **form** element is a container element in HTML where you can put controls

- text boxes
- password boxes
- checkbox, radio buttons
- select boxes - and more..

Lets create a simple in the **public** directory of our new site, with text boxes for first and last name of a person

HTML Forms



- We'll come back to the text boxes…
- The button (type = submit) causes an HTTP request to be made.
- The request is sent to whatever URL is specified in the action attribute of the form.
- The method of the request could be GET or POST.
  - GET will show a query string in the address bar

http://localhost:4000/save.html?first=Scott&last=Frees

The query string

http://localhost:4000/save.html?first=Scott&last=Frees

- The URL is in two parts - before the **?** is the typical resource we've seen before.
- After the **?** is a series of name/value pairs, each separated by **&**.
- Note - illegal URL characters (like spaces) are replaced
- These are called **parameters**, and they are accessible in the HTTP request object!

Getting the query params

```
var url = require('url');
var url_parts = url.parse(req.url, true);
var query = url_parts.query;

for ( q in query ) {
    console.log(q + " -> " + query[q]);
}
```

The url module in Node lets us pull the query parameters out of the URL and use them like an array.

POST

- GET is a good method if you want the destination to be bookmarkable, but it has its limits.
  - The number/size of query parameters is limited
  - Sensitive information is wide open

- POST sends the query parameters as the HTTP request's message body.

POST Data - Node stuff...

```
if(req.method=='POST')  {
    var body = "";
    req.on('data', function (chunk) {
        body += chunk;
    });
    req.on('end', function() {
        qs = require('querystring');
        var post =  qs.parse(body);
        for ( q in post ) {
            console.log(q + " -> " + post[q]);
        }
    });
}
```

The **on** function is registering functions to handle events.

The req object is an **event emitter**.

See Node.js text, Chapter 5 (or just wait a few weeks)

Form Controls

Those text boxes are **input** elements.

Input elements have several supported attributes:

- **name**: the name of the control, should be unique within the form. Will be the name sent in the query string or post data.
- **id**: all HTML element can have ID - your form elements typically should match name and id (but not always)
- **type**: Next Slide
- **value**: Initial value (optional)

Only elements with a name attribute will be submitted when the form is submitted.

input types

There are several valid **type** values

- text: a simple text box
- password: masks characters
- hidden: the value will be sent in the query string, but the control is not visible

- input elements are always **empty**
- input elements typically support the following additional attributes:

  placeholder, disabled, readonly, required

Multi-line text boxes

The **textarea** element supports multi-line text entry.

```
<textarea name="comments">
    some comments...
</textarea>
```

- Unlike input, textarea uses the element content for the value.
- It is submitted the same way as single-line inputs
- Supports many of the same fields
- You can specify **rows** and **cols** attributes to control size

Checkboxes

```
<input type="checkbox" name="chkbx1"/>
```

Checkboxes don't have a label associated with them unless you add a <label> element.

The checked state is controlled by a boolean attribute - **checked**

```
<input type="checkbox" name="chkbx2" checked/>
```

Checkboxes - Server-Side

When a checkbox is checked, the name value pair is sent to the server on form submission

```
<input type="checkbox" name="chkbx1" checked/>
```

chkbx1=on sent to server

If its not checked, **nothing is sent to server**.

This can cause a bit of an issue server-side - don't forget this!

Radios

- Radio buttons can present multiple options to users.
- Like checkboxes, you must explicitly add text through a label or some other HTML text element.
- Radio buttons are grouped together by a common **name** attribute (id should be unique)

```
<p><input type="radio" name="radios" id="r1" value="a">A</p>
<p><input type="radio" name="radios" id="r2" value="b">B</p>
<p><input type="radio" name="radios" id="r3" value="c">C</p>
```

You can specify the **checked** attribute to make one radio button selected by default.

Select Boxes

Select boxes can work in most of the same situations as radios

```
<select name="select" id="select">
    <option value="a">A</option>
    <option value="b">B</option>
    <option value="c">C</option>
</select>
```

Buttons

A form typically will have at least one button, which allows the user to submit the form

```
<form action="save.html" method="post">
        <input type="text" name="name" value="test"/>
        <button type="submit">Save</button>
        <button type="reset">Reset</button>
        <button type="button">Cancel</button>
</form>
```

Always specify the type!

Buttons

- A button of type “button” appears useless… but its not!
  - We will soon be writing client-side JavaScript
  - We will be able to attach JavaScript functions to the button
  - We’ll also write JavaScript that can submit a form without the user ever clicking a submit button.

- Buttons can be heavily styled with CSS

HTML5 input types

There is a lot of input that is improved with UI controls. Recently, browsers have begun to implement the full set of HTML5 controls

color, date, datetime, datetime-local, email, month, number, range, search, tel, time, url, week

The great part about these is that they work well on mobile devices, tablets, etc.

http://diveintohtml5.info/forms.html is a fantastic resource for HTML5 input elements

HTML5 input types

In addition to the placeholder, autofocus, required attributes, the HTML5 input types drastically improve your life.

But - not all are supported. Always consult a resource to see if you are using something that is generally supported...

http://caniuse.com/#feat=forms

The good news is that unknown types will default to "text"

Next up - dynamic data

Now that we have the data on the server, we can use it to customize the HTML we send to the browser.

To do this, we will use a *templating engine* called EJS.

We will embed *server-side* javascript in HTML-like templates