

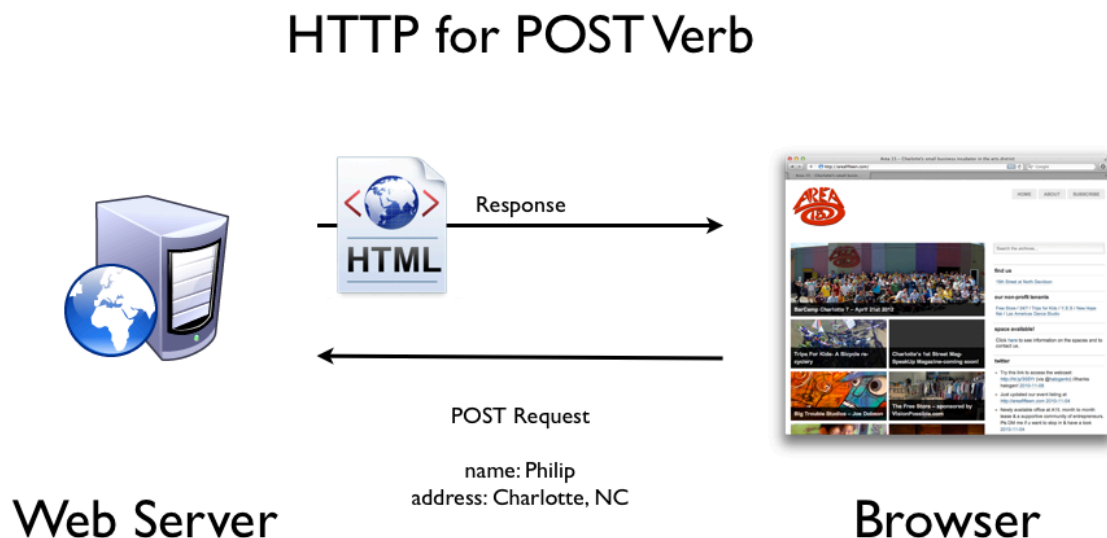
Interacting with the Server

Introduction

So you have a basic web server up and running now - you have been able to serve up pages and build up your controller so you can handle different paths.

Now we need to move up to the next level and we need to be able accept posts back to the server from the browser.

Remember we said that we have these things called VERBS in the HTTP protocol. These verbs allow you to tell the webserver slightly different things, so far we have been working on the GET verb - ie. the browsers asks to GET a page from the web server. Next up we are going to work on the POST. This is where the browser posts content back.



Browsers can use many ways to POST information form a page back to a server, lets start with the simplest way which is Forms.

Using Forms

Basically browsers know how to handle form tags and the input tags that go inside them - process them to build a form you can type into - then the submit input allows you to POST this information back to the server.

A form is a pretty simple set of tags, though as you learn in the future there are lots of form controls.

```
<form method="post" action="/sayHello">
  Your name:<br/>
  <input type="text" name="firstname" /><br />
  <input type="submit" value="Submit" />
</form>
```

The HTTP verb

The Path to go to

Have a button to send it

Capture any fields we want

Basically they are just HTML tags - the form *wraps* the input elements, and we use attributes on the form tag to tell it that we want to use the POST verb and also the path that we want to send the information to.

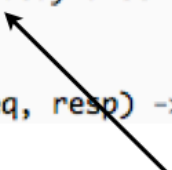
We have set-up the input and types that we want to use - your HTML tags cheat sheet has lots of these, and also we will input a submit button which will allow the form to POST to the server. We need to take note of the names for the fields that we create since we will reference these as variables on our server when we process the POST request.

Handling the POST on the Server

Now we need the server to be able to handle taking a post from a form - well since we are using the express web framework there is actually a little helper we can add to the application we created to specifically look for form variables in the post body and make them accessible as properties on the request object.

This little helper is set-up in our app.coffee.

```
10 # Setup Static Files
11 app.use express.static(__dirname + '/public')
12
13 # Setup Assets
14 app.use require('connect-assets')()
15 app.use express.bodyParser()
16
17 # App Routes
18 app.get '/', (req, resp) ->
```



This adds the a part of the express tool
to handle the form post

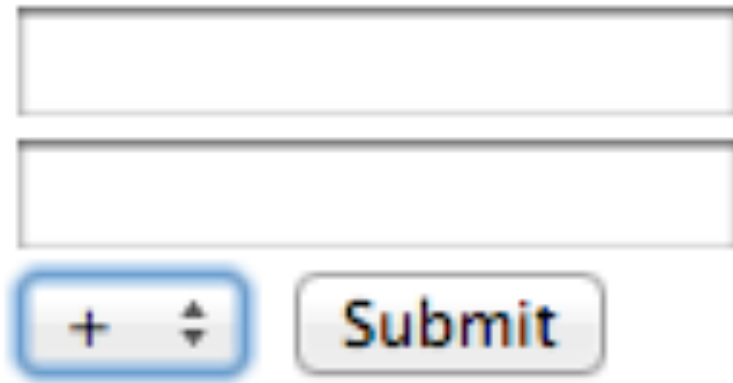
Next up we need to add a route to handle the post - on our form we set the action attribute on the form tag to be “/sayHello”, this means that the form is going to POST to that path - so we can add a handler. Now note that before we used app.get since we were handling a get request - now we are handling a post request so we use app.post.

```
19 resp.render 'index.eco', title: 'This is where we start your new p
20
21 app.post '/sayHello', (req, resp) ->
22   resp.render 'sayHello.eco', firstname: req.body.firstname
23
24 # Listen
25 app.listen process.env.C9_PORT -> console.log 'Now running'
```

Also since we put our express.bodyParser() into our app we are now able to look for the input we posted as req.body.XXXX where XXXX is the name of the input on the form.

Exercise

Build a simple calculator using the POST and the server to do the calculations



The image shows a simple web form for a calculator. It consists of two empty text input fields stacked vertically. Below the second input field, there is a button with a plus sign and a divide sign, and another button labeled "Submit".

Bear in mind a few things - the names you give the input tags will be the names you end up with in the body.req, ie. if you have

```
<input type="text" name="firstNumber"/>
```

then you will end up with

req.body.firstNumber in your controller.

Also it is important to remember that everything that comes from a form is a String type - a String is what we called a collection of characters. This means if you have your two inputs called firstNumber and secondNumber and do

```
result = firstNumber + secondNumber
```

it will end up “concatenating” the two String together - so if there values werevm 1 and 2 you would end up with 12. We need to turn them into Numbers so that the arithmetic operations work properly by doing the following.

```
result = Number(firstNumber) + Number(secondNumber)
```

Javascript is everywhere

Our server side code is quite limited but you can see that we are actually building applications now - these basic steps are absolutely no different from the steps a professional developer would take - and in fact the tools you are using now are the state-of-the-art in web development.

Now we need to go back and take a look at how we can add code in the browser, the ability to build complex applications now is the fine art of understanding how to build server side logic and then also how you can control the browser itself.

So how do you use Javascript in the browser?

Well with the script tag?

```
<script>
    alert("This is Javascript");
```

```
</script>
```

And how do you get the contents of an input - by looking it up by its id, consider the following HTML:

```
<body>
```

```
<script>
    function sayHello() {
        var myTextField = document.getElementById('firstname');
        if(myTextField.value != "")
            alert("Hello " + myTextField.value);
        else
            alert("Please enter your name");
    }
</script>
```

```
<input id="firstname" type="text"/>
```

```
<a onclick="sayHello()"/>
```

Give it a go in the browser on your index page. Also remember you can set a value on a text field using

```
myTextField.value = "12"
```

Exercise

Now rebuild your calculator - but this time don't POST the numbers back to the server - put the logic in the browser.