EJS

SENG 4640
Software Engineering for Web Apps
Winter 2023

Sina Keshvadi
Thompson Rivers University

Review

- Node.js and Express allow us to build server-side web apps in JavaScript
- HTTP Requests and Responses are represented as JavaScript objects
- We can get data from user either via the URL or through form submissions

We've been building up our knowledge of node in Express by looking at the core request and response objects, and then how we can have different functionality for different requests.

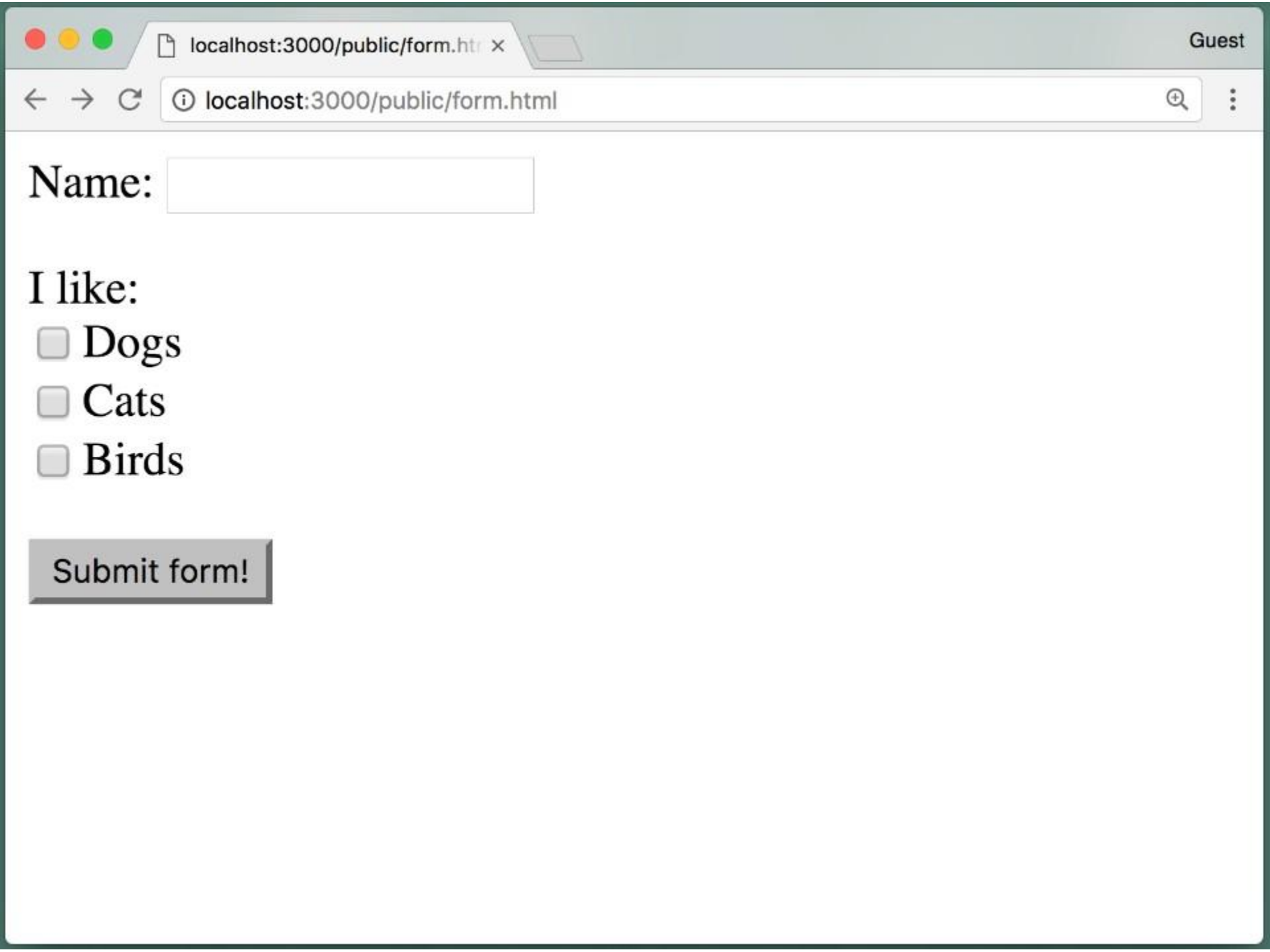
We've been building up our knowledge of node in Express by looking at the core request and response objects, and then how we can have different functionality for different requests.

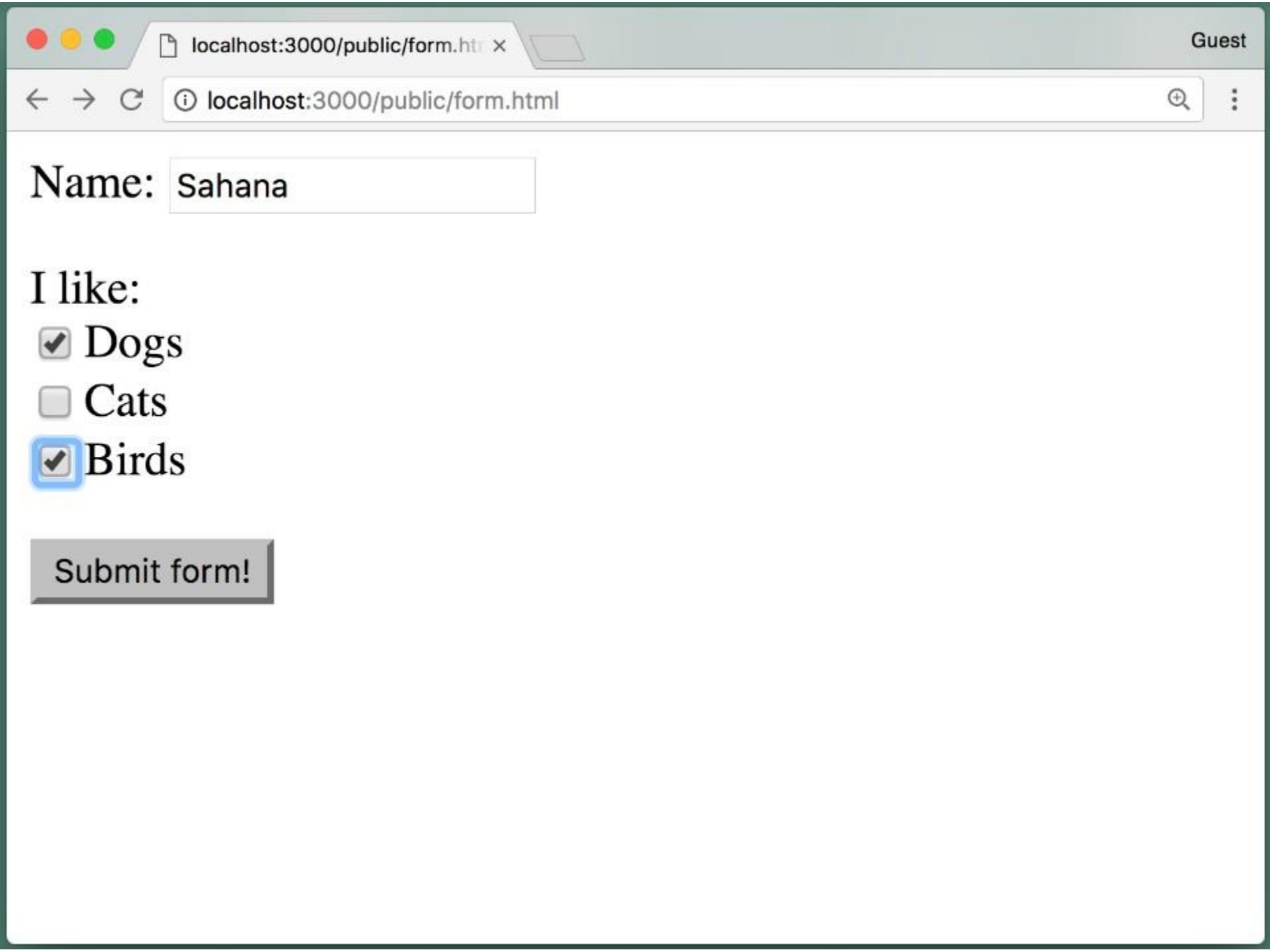
And now, we can get data from the user.

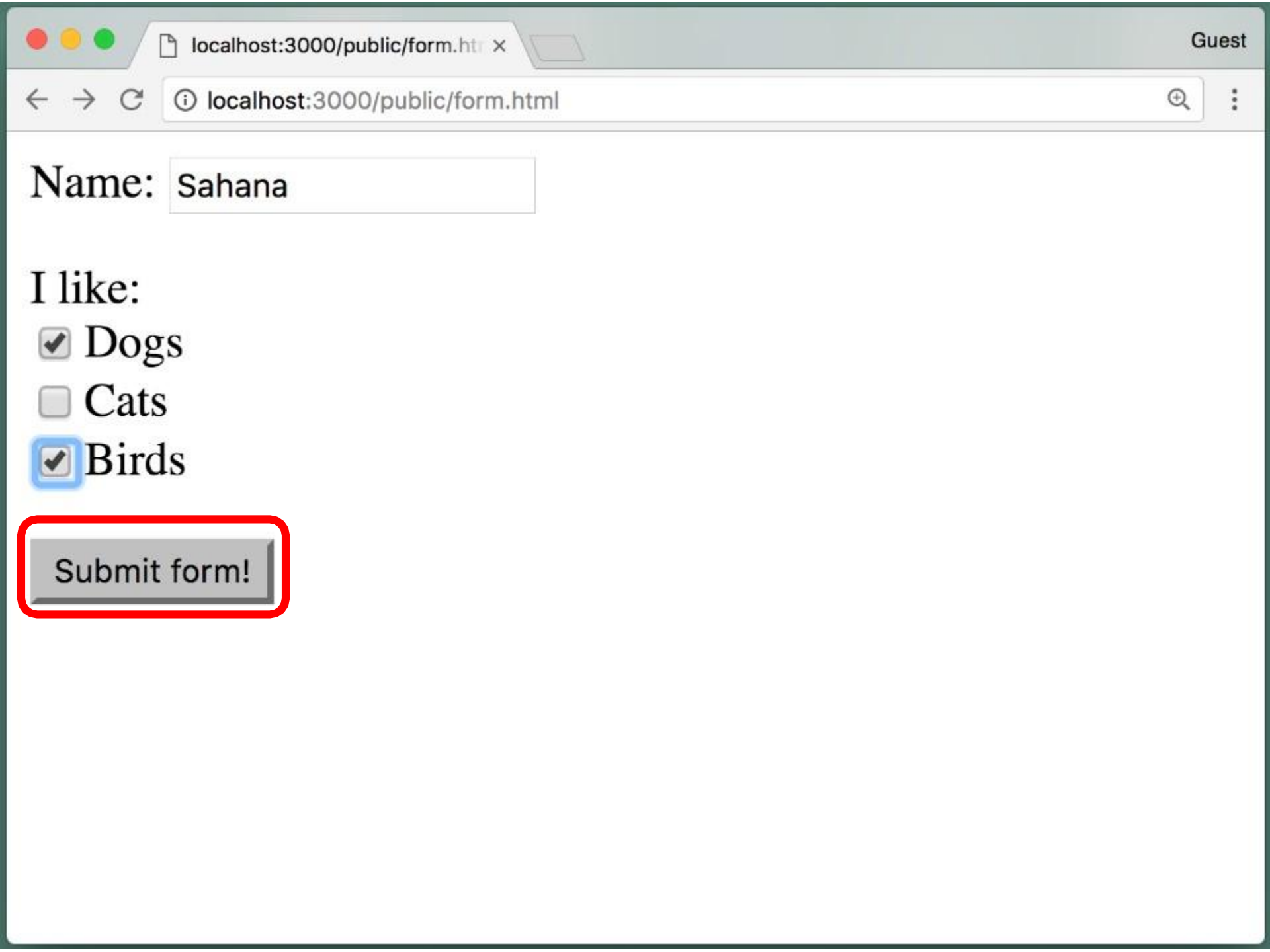
We've been building up our knowledge of node in Express by looking at the core request and response objects, and then how we can have different functionality for different requests.

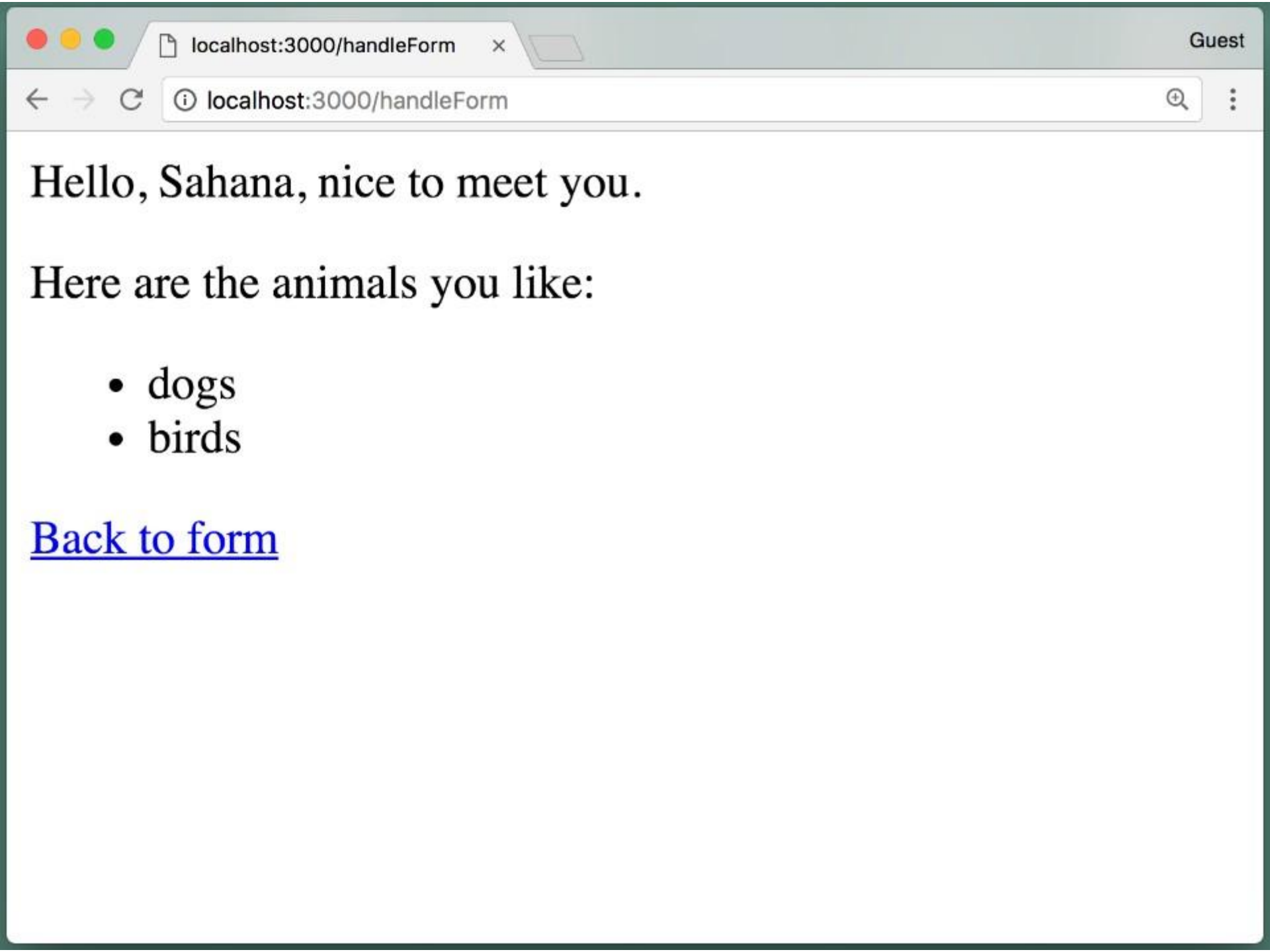
And now, we can get data from the user.

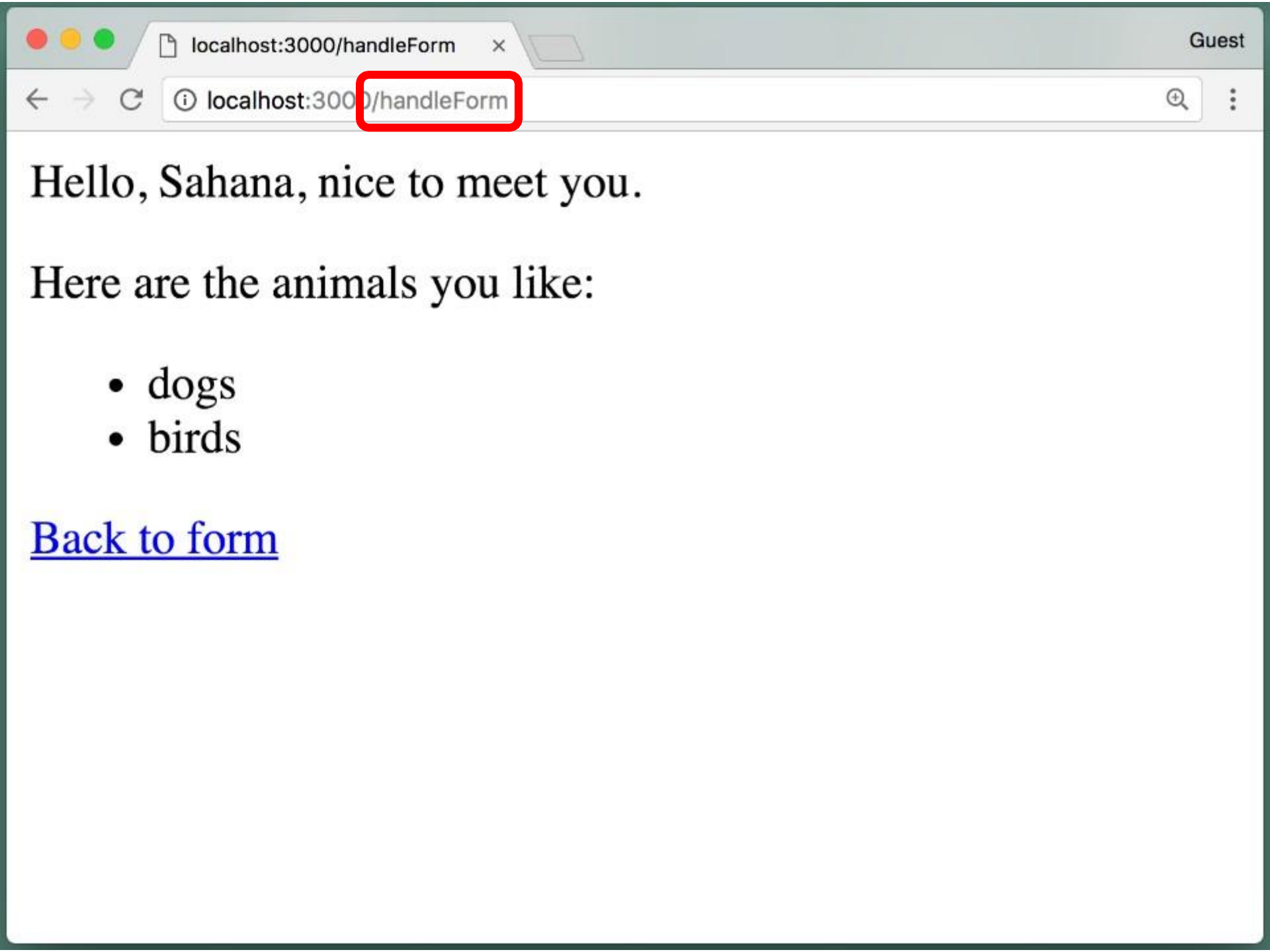
In the next few lectures, we'll start to put all these together into a full fledged web application, where we're serving content based on different functionality in our app.


localhost:3000/public/form.ht
Guest
localhost:3000/public/form.html
Name:
I like:
Dogs
Cats
Birds
Submit form!


localhost:3000/public/form.ht
Guest
localhost:3000/public/form.html
Name:
Sahana
I like:
Dogs
Cats
Birds
Submit form!


localhost:3000/public/form.ht
Guest
localhost:3000/public/form.html
Name: Sahana
I like:
Dogs
Cats
Birds
Submit form!


localhost:3000/handleForm
Guest
localhost:3000/handleForm
Hello, Sahana, nice to meet you.
Here are the animals you like:
dogs
birds
Back to form


localhost:3000/handleForm
Guest
localhost:3000/handleForm
Hello, Sahana, nice to meet you.
Here are the animals you like:
dogs
birds
Back to form

```
var bodyParser = require('body-parser');
app.use(bodyParser.urlencoded({ extended: true }));

app.use('/handleForm', (req, res) => {
   var name = req.body.username;
   var animals = [].concat(req.body.animal);
   res.type('html').status(200);
   res.write('Hello, ' + name + ', nice to meet you.');
   res.write('<p>Here are the animals you like:');
   res.write('<ul>');
   animals.forEach( (animal) => { res.write('<li>'
      + animal + '</li>');
   });
   res.write('</ul>');
   res.write("<a href='/public/form.html'>" + "Back to
                                              form</a>");
   res.end();
});
```

use the body-parser middleware, because the data coming from the user is part of the form that was submitted using an HTTP post request.

```
var bodyParser = require('body-parser');
app.use(bodyParser.urlencoded({ extended: true }));

app.use('/handleForm', (req, res) => {
   var name = req.body.username;
   var animals = [].concat(req.body.animal);
   res.type('html').status(200);
   res.write('Hello, ' + name + ', nice to meet you.');
   res.write('<p>Here are the animals you like:');
   res.write('<ul>');
   animals.forEach( (animal) => { res.write('<li>'
      + animal + '</li>');
   });
   res.write('</ul>');
   res.write("<a href='/public/form.html'>" + "Back to
                                              form</a>");
   res.end();
});
```

Username from our form.
And remember the body parser middleware, would put the form data into the body property of the request.

```
var bodyParser = require('body-parser');
app.use(bodyParser.urlencoded({ extended: true }));

app.use('/handleForm', (req, res) => {
   var name = req.body.username;
   var animals = [].concat(req.body.animal);
   res.type('html').status(200);
   res.write('Hello, ' + name + ', nice to meet you.');
   res.write('<p>Here are the animals you like:');
   res.write('<ul>');
   animals.forEach( (animal) => { res.write('<li>'
      + animal + '</li>');
   });
   res.write('</ul>');
   res.write("<a href='/public/form.html'>" + "Back to
                                              form</a>");
   res.end();
});
```

we want to be an array of the animals. So here's a little JavaScript trick, start with an empty array and then use the .concat function to concatenate whatever is in the body of the request with that empty array.

```
var bodyParser = require('body-parser');
app.use(bodyParser.urlencoded({ extended: true }));

app.use('/handleForm', (req, res) => {
   var name = req.body.username;
   var animals = [].concat(req.body.animal);
   res.type('html').status(200);
   res.write('Hello, ' + name + ', nice to meet you.');
   res.write('<p>Here are the animals you like:');
   res.write('<ul>');
   animals.forEach( (animal) => { res.write('<li>'
      + animal + '</li>');
   });
   res.write('</ul>');
   res.write("<a href='/public/form.html'>" + "Back to
                                              form</a>");
   res.end();
});
```

Keep in mind that all of the code we're seeing here is server-side, it's not client-side. All of this code is running in the server.

- This code works fine but maybe we want to do something to clean up all of this.

- This code works fine but maybe we want to do something to clean up all of this.
- Often we want to separate format from functionality.

- This code works fine but maybe we want to do something to clean up all of this.
- Often we want to separate format from functionality.
- That is we want to separate the appearance of what we're producing, from the code that produces it.

- This code works fine but maybe we want to do something to clean up all of this.
- Often we want to separate format from functionality.
- That is we want to separate the appearance of what we're producing, from the code that produces it.
- In this case we want to separate the HTML from the JavaScript.

- This code works fine but maybe we want to do something to clean up all of this.
- Often we want to separate format from functionality.
- That is we want to separate the appearance of what we're producing, from the code that produces it.
- In this case we want to separate the HTML from the JavaScript.
- In order to separate the JavaScript from the HTML, in Express we can us a tool called **EJS**, or EmbeddedJS, or embedded JavaScript.

- This code works fine but maybe we want to do something to clean up all of this.
- Often we want to separate format from functionality.
- That is we want to separate the appearance of what we're producing, from the code that produces it.
- In this case we want to separate the HTML from the JavaScript.
- In order to separate the JavaScript from the HTML, in Express we can us a tool called EJS, or EmbeddedJS, or embedded JavaScript.
- EJS is a view engine that combines data or static content, with embedded JavaScript, and executes the JavaScript to generate HTML.

- This code works fine but maybe we want to do something to clean up all of this.
- Often we want to separate format from functionality.
- That is we want to separate the appearance of what we're producing, from the code that produces it.
- In this case we want to separate the HTML from the JavaScript.
- In order to separate the JavaScript from the HTML, in Express we can us a tool called EJS, or EmbeddedJS, or embedded JavaScript.
- EJS is a view engine that combines data or static content, with embedded JavaScript, and executes the JavaScript to generate HTML.
- This means that our HTML can be rendered dynamically on the server, and then sent back as content to the browser.

What is EJS?

- EJS, or **EmbeddedJS**, is a **view engine** that uses data and embedded JavaScript to produce HTML

- This allows webpages to be developed statically and rendered dynamically server-side

- EmbeddedJS is a package that can be installed with the command: `npm install ejs`

Using EJS in an Express app

- Set EJS as the default rendering method in your app with **`app.set('view engine', 'ejs');`**

```
var express = require('express');
var app = express();

app.set('view engine', 'ejs');

app.get('/', (req, res) => {
  res.render('welcome', {username:
  'CandyLover'});
});
```

There are many different view engines that we can use, and ejs is just one of them, and in our app, we would set the view engine to ejs.

Using EJS in an Express app

- Set EJS as the default rendering method in your app with **`app.set('view engine', 'ejs');`**

```
var express = require('express');
var app = express();

app.set('view engine', 'ejs');

app.get('/', (req, res) => {
  res.render('welcome', {username: 'CandyLover'});
});
```

Then, rather than using the response, write, or send function, we would use the render function.

Using EJS in an Express app

- Set EJS as the default rendering method in your app with **`app.set('view engine', 'ejs');`**

```
var express = require('express');
var app = express();

app.set('view engine', 'ejs');

app.get('/', (req, res) => {
  res.render('welcome', {username:'CandyLover'});
});
```

The render function takes the name of the EJS file as it's first argument. By default the ejs files should go in the **views** sub directory or folder, and we don't need to say .ejs at the end because it's understood.

Using EJS in an Express app

- Set EJS as the default rendering method in your app with **`app.set('view engine', 'ejs');`**

```
var express = require('express');
var app = express();

app.set('view engine', 'ejs');

app.get('/', (req, res) => { res.render('welcome',
  {username: 'CandyLover'});
});
```

If we want to pass arguments to the ejs file, we do that using JavaScript objects.

So now let's see what an .ejs file looks like.

Writing EJS files

- A .ejs file is just an HTML file that has JavaScript code embedded in it
- Anything between **<%=** and **%>** tags will be evaluated and incorporated into the HTML
- By default, the .ejs files should be in the **views/** subdirectory of the Express project

```
<!-- This is views/welcome.ejs -->

<!DOCTYPE html>
<html>
<body>

<h1>Welcome, <%= username %>!</h1>

</body>
</html>
```

EJS and JavaScript

- EJS will execute any JavaScript that appears between **<%** and **%>** tags when generating the HTML page on the server

```
res.render('welcome', {username: 'CandyLover', isAdmin: true});
```

```
<!DOCTYPE html>
<html>
<body>

<h1>Welcome, <%= username %>!</h1>
<% if (isAdmin) { %>
    <p> Remember to check your email every 24 hours! </p>
<% } %>

</body>
</html>
```

render function on the response uses welcome.ejs and would send the argument that includes two key value pairs

EJS and JavaScript

- EJS will execute any JavaScript that appears between **<%** and **%>** tags when generating the HTML page on the server

```
res.render('welcome', {username: 'CandyLover', isAdmin: true});
```

```
<!DOCTYPE html>
<html>
<body>

<h1>Welcome, <%= username %>!</h1>
<% if (isAdmin) { %>
    <p> Remember to check your email every 24 hours! </p>
<% } %>

</body>
</html>
```

evaluate username using <% and %>

EJS and JavaScript

- EJS will execute any JavaScript that appears between **<%** and **%>** tags when generating the HTML page on the server

```
res.render('welcome', {username: 'CandyLover', isAdmin: true});
```

```
<!DOCTYPE html>
<html>
<body>

<h1>Welcome, <%= username %>!</h1>
<% if (isAdmin) { %>
    <p> Remember to check your email every 24 hours! </p>
<% } %>

</body>
</html>
```

javascript using <% and %>

EJS and JavaScript

- EJS will execute any JavaScript that appears between **<%** and **%>** tags when generating the HTML page on the server

```
res.render('welcome', {username: 'CandyLover', isAdmin: true});
```

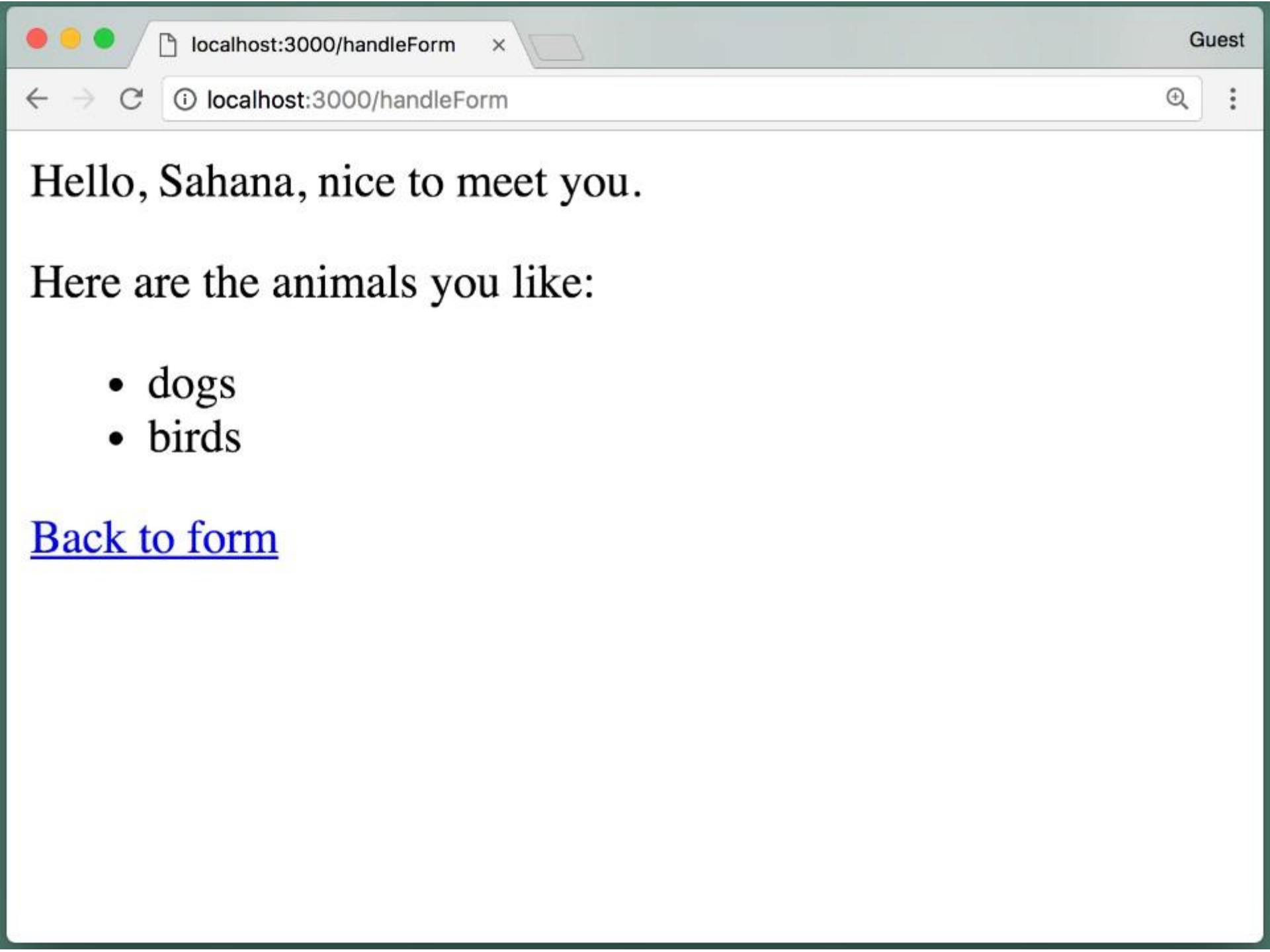
```
<!DOCTYPE html>
<html>
<body>

<h1>Welcome, <%= username %>!</h1>
<% if (isAdmin) { %>
    <p> Remember to check your email every 24 hours! </p>
<% } %>

</body>
</html>
```

use EJS to separate the JavaScript and the HTML

Let's go back to our motivating example.
Whereas part of handling the form, we want to echo back the data that the user sent but we wanna do that without tangling up all of our JavaScript and HTML within the index.js file.


localhost:3000/handleForm
Guest
localhost:3000/handleForm
Hello, Sahana, nice to meet you.
Here are the animals you like:
dogs
birds
Back to form

```
var bodyParser = require("body-parser");
app.use(bodyParser.urlencoded({ extended: true }));

app.use("/handleForm", (req, res) => {
 var name = req.body.username;
 var animals = [].concat(req.body.animal);
 res.type("html").status(200);
 res.write("Hello, " + name + ", nice to meet you.");
 res.write("<p>Here are the animals you like:");
 res.write("<ul>");
 animals.forEach((animal) => {
   res.write("<li>" + animal + "</li>");
 });
 res.write("</ul>");
 animals.forEach((animal) => {
   res.write("<li>" + animal + "</li>");
 });
 res.write("</ul>");
 res.write("<a href='/public/form.html'>" + "Back to form </a>");
 res.end();
});
```

And all of this code is the code we'd like to remove.

```
var bodyParser = require("body-parser");
app.use(bodyParser.urlencoded({ extended: true }));

app.use("/handleForm", (req, res) => {
 var name = req.body.username;
 var animals = [].concat(req.body.animal);
 res.type("html").status(200);
 res.write("Hello, " + name + ", nice to meet you.");
 res.write("<p>Here are the animals you like:");
 res.write("<ul>");
 animals.forEach((animal) => {
   res.write("<li>" + animal + "</li>");
 });
 res.write("</ul>");
 animals.forEach((animal) => {
   res.write("<li>" + animal + "</li>");
 });
 res.write("</ul>");
 res.write("<a href='/public/form.html'>" + "Back to form </a>");
 res.end();
});
```

All this code is not really the code for the logic of our program.
But is really just the rendering of the HTML.

```
var bodyParser = require("body-parser");
app.use(bodyParser.urlencoded({ extended: true }));

app.use("/handleForm", (req, res) => {
 var name = req.body.username;
 var animals = [].concat(req.body.animal);
 res.render("showAnimals", {name:name, animals:animals});
});
```

we move that entirely and replace it with the render function.
the ejs file, that we want to render, is showanimals.

```
var bodyParser = require("body-parser");
app.use(bodyParser.urlencoded({ extended: true }));

app.use("/handleForm", (req, res) => {
 var name = req.body.username;
 var animals = [].concat(req.body.animal);
 res.render("showAnimals", {name:name, animals:animals});
});
```

```
<!-- This is views/showAnimals.ejs -->
Hello, <%= name %>, nice to meet you.
<p>Here are the animals you like:
<ul>
<% animals.forEach( (animal) => { %>
   <li> <%= animal %> </li>
<% }); %>
</ul>
<a href='/public/form.html'>Back to form</a>
```

Note: showanimals.ejs would need to be in the views subdirectory.

```
var bodyParser = require("body-parser");
app.use(bodyParser.urlencoded({ extended: true }));

app.use("/handleForm", (req, res) => {
 var name = req.body.username;
 var animals = [].concat(req.body.animal);
 res.render("showAnimals", {name:name, animals:animals});
});
```

```
<!-- This is views/showAnimals.ejs -->
Hello, <%= name %>, nice to meet you.
<p>Here are the animals you like:
<ul>
<% animals.forEach( (animal) => { %>
   <li> <%= animal %> </li>
<% }); %>
</ul>
<a href='/public/form.html'>Back to form</a>
```

name came as one of the arguments that was passed to the render function.

```
var bodyParser = require("body-parser");
app.use(bodyParser.urlencoded({ extended: true }));

app.use("/handleForm", (req, res) => {
 var name = req.body.username;
 var animals = [].concat(req.body.animal);
 res.render("showAnimals", {name:name, animals:animals});
});
```

```
<!-- This is views/showAnimals.ejs -->
Hello, <%= name %>, nice to meet you.
<p>Here are the animals you like:
<ul>
<% animals.forEach( (animal) => { %>
   <li> <%= animal %> </li>
<% }); %>
</ul>
<a href='/public/form.html'>Back to form</a>
```

iterate over all of the animals that are in the array, using the forEach function.

```
var bodyParser = require("body-parser");
app.use(bodyParser.urlencoded({ extended: true }));

app.use("/handleForm", (req, res) => {
 var name = req.body.username;
 var animals = [].concat(req.body.animal);
 res.render("showAnimals", {name:name, animals:animals});
});
```

```
<!-- This is views/showAnimals.ejs -->
Hello, <%= name %>, nice to meet you.
<p>Here are the animals you like:
<ul>
<% animals.forEach( (animal) => { %>
   <li> <%= animal %> </li>
<% }); %>
</ul>
<a href='/public/form.html'>Back to form</a>
```

Summary

- **EJS** allows webpages to be developed statically and rendered dynamically server-side

- An Express app can generate and send the HTML from a .ejs file using the Response's `render` function

- EJS will execute any JavaScript that appears between `<%` and `%>` tags when generating the HTML page on the server