Getting User Data

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 display content based on different routes through the app

In general, there are two different mechanisms for getting data from the users via the HTTP request.

Getting Data from Users: HTTP Requests

- **Query parameters**
  - Key/value pairs that are part of the URL
  - Can be part of a static URL or be generated by an HTML form using the “GET” method

- **POST data**
  - Key/value pairs that are included in the body of the HTTP request
  - Result from an HTML form using the “POST” method

Let's see the idea of the request including a query in a little more detail.

Request Object Query Properties

- An HTTP Request object can include **query** properties that come from the URL

http://localhost:3000/?name=Lydia&location=United+States

```
app.use('/', (req, res) => {

  var query = req.query;

  console.log(query);

  var name = query.name;   // 'Lydia'
  var location = query.location; // 'United States'

  var length = Object.keys(query).length; // 2
  res.send('Hello World!');
 });
```

```
The middleware function will take the request and access its query property.
```

Request Object Query Properties

- An HTTP Request object can include **query** properties that come from the URL

http://localhost:3000/?name=Lydia&location=United+States

```
app.use('/', (req, res) => {

  var query = req.query;

  console.log(query);

  var name = query.name;   // 'Lydia'
  var location = query.location; // 'United States'

  var length = Object.keys(query).length; // 2
  res.send('Hello World!');
 });
```

The query always contains key/value pairs that start with a question mark, and then use the ampersand, or & sign, in between the different key/value pairs.

Request Object Query Properties

- An HTTP Request object can include **query** properties that come from the URL

http://localhost:3000/?name=Lydia&location=United+States

```
app.use('/', (req, res) => {

  var query = req.query;

  console.log(query);

  var name = query.name;   // 'Lydia'
  var location = query.location; // 'United States'

  var length = Object.keys(query).length; // 2
  res.send('Hello World!');
 });
```

```
log the query to the console to see that this is just a JavaScript object.
```

Request Object Query Properties

- An HTTP Request object can include **query** properties that come from the URL

http://localhost:3000/?name=Lydia&location=United+States

```
app.use('/', (req, res) => {

  var query = req.query;

  console.log(query);

  var name = query.name;   // 'Lydia'
  var location = query.location; // 'United States'

  var length = Object.keys(query).length; // 2
  res.send('Hello World!');
 });
```

Request Object Query Properties

- An HTTP Request object can include **query** properties that come from the URL

http://localhost:3000/?name=Lydia&location=United+States

```
app.use('/', (req, res) => {

  var query = req.query;

  console.log(query);

  var name = query.name;   // 'Lydia'
  var location = query.location; // 'United States'

  var length = Object.keys(query).length; // 2
  res.send('Hello World!');
 });
```

```
If we want to know how many keys there are within the query
```

Express, also, allows us to use what are known as **parameterized URLS**, where instead of using the key/value pairs separated by the question mark and ampersand, we can include them right in the URI.

Request Object Parameters

- An HTTP Request object can include **param** properties that come from a parameterized URL

http://localhost:3000/name/Lydia/location/United States

```
 app.use('/name/:userName/location/:userLocation',
      (req, res) => {

    var params = req.params;
    console.log(params);

    var name = params.userName;  // 'Lydia'
    var location = params.userLocation; // 'United States'

    var length = Object.keys(params).length; // 2
    res.send('Hello World!');
});
```

Request Object Parameters

- An HTTP Request object can include **param** properties that come from a parameterized URL

http://localhost:3000/name/Lydia/location/United States

```
 app.use('/name/:userName/location/:userLocation',
      (req, res) => {

    var params = req.params;
    console.log(params);

    var name = params.userName;   // 'Lydia'
    var location = params.userLocation; // 'United States'

    var length = Object.keys(params).length; // 2
    res.send('Hello World!');
});
```

Request Object Parameters

- An HTTP Request object can include **param** properties that come from a parameterized URL

http://localhost:3000/name/Lydia/location/United States

```
 app.use('/name/:userName/location/:userLocation',
      (req, res) => {

    var params = req.params;
    console.log(params);

    var name = params.userName;  // 'Lydia'
    var location = params.userLocation; // 'United States'

    var length = Object.keys(params).length; // 2
    res.send('Hello World!');
});
```

```
rather than accessing req.query, here we're accessing req.params.
```

Request Object Parameters

- An HTTP Request object can include **param** properties that come from a parameterized URL

http://localhost:3000/name/Lydia/location/United States

```
 app.use('/name/:userName/location/:userLocation',
      (req, res) => {

    var params = req.params;
    console.log(params);

    var name = params.userName;   // 'Lydia'
    var location = params.userLocation; // 'United States'

    var length = Object.keys(params).length; // 2
    res.send('Hello World!');
});
```

Request Object Parameters

- An HTTP Request object can include **param** properties that come from a parameterized URL

http://localhost:3000/name/Lydia/location/United States

```
 app.use('/name/:userName/location/:userLocation',
      (req, res) => {

    var params = req.params;
    console.log(params);

    var name = params.userName;   // 'Lydia'
    var location = params.userLocation; // 'United States'

    var length = Object.keys(params).length; // 2
    res.send('Hello World!');
});
```

Open this link:
http://localhost:3000/name/Lydia/location/United%20States

Note: %20 is space
(full list https://www.w3schools.com/tags/ref_urlencode.ASP)

We've seen how to get user data out of the URL itself through the query or parameters.
But what about forms?

But what happens when the user submits the form?

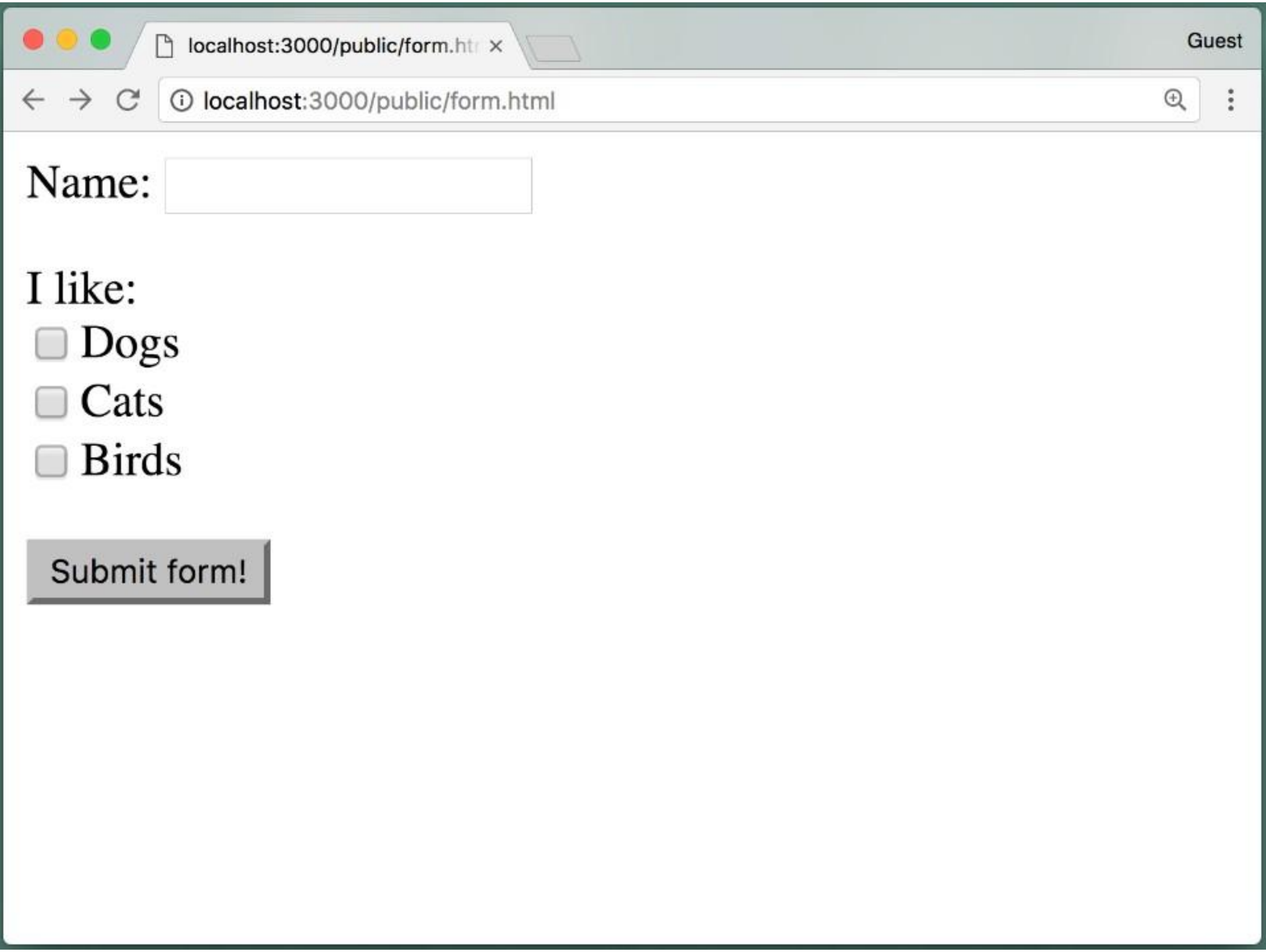
How does the data get to the server?

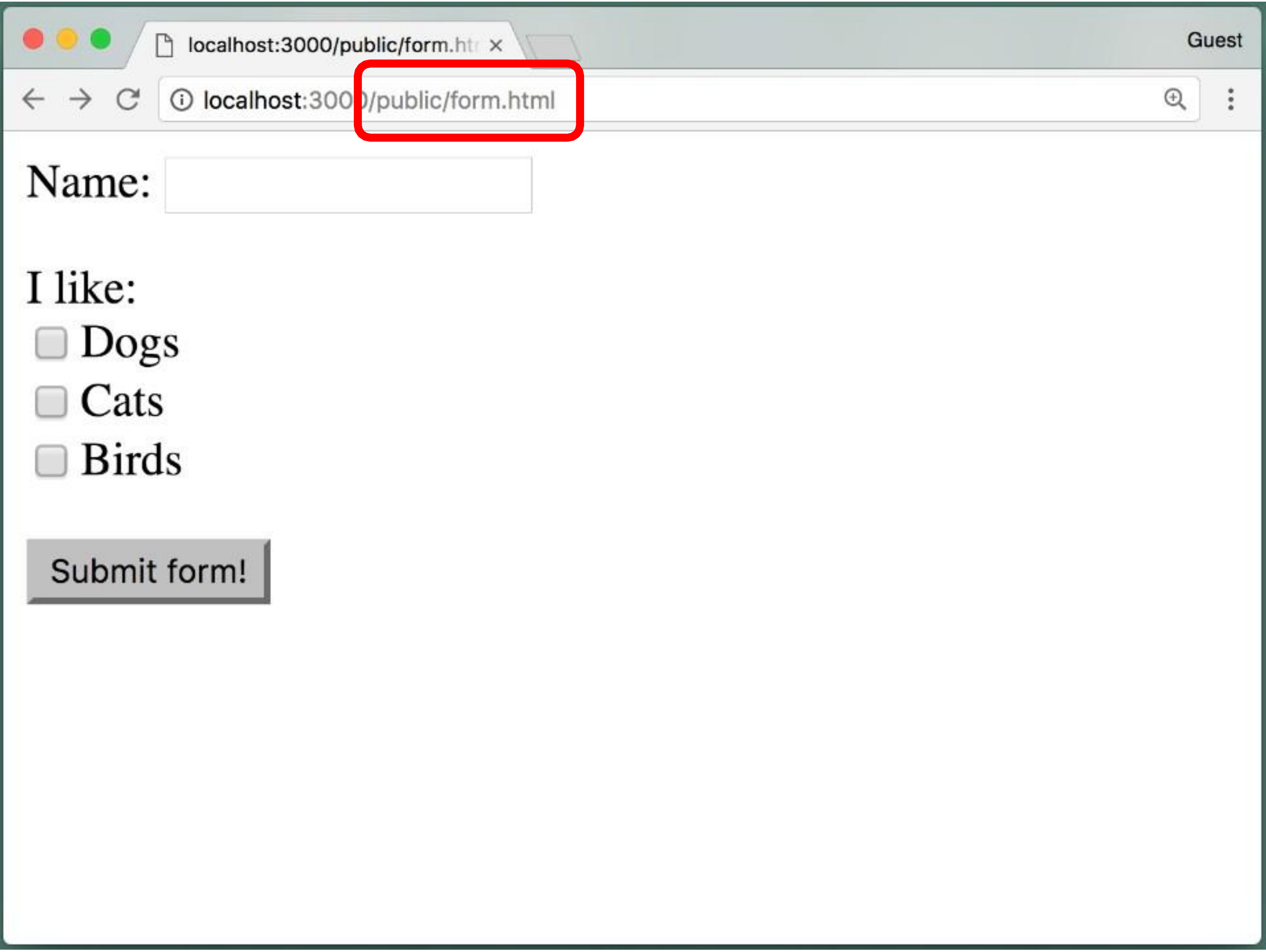
How can our Express app use that data to then render content?

HTML Forms

- Forms allow users to enter or select data, e.g. via input boxes, checkboxes, radio buttons, etc.

- The form specifies the **action** and **method** that result when the user chooses to **submit** the form
  - Action: the URL to be requested
  - Method: the HTTP Request “verb,” e.g. GET or POST


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


localhost:3000/public/form.htr
Guest
localhost:3000/public/form.html

Name:

I like:

- Dogs
- Cats
- Birds

```
<--! Form.html file -->
<html>
<body>

<form action="/handleForm" method="post">

Name: <input name="username">
<p>
I like:<br>
<input type=checkbox name="animal" value="dogs">Dogs <br>
<input type=checkbox name="animal" value="cats">Cats <br>
<input type=checkbox name="animal" value="birds">Birds <br>
<p>
<input type=submit value="Submit form!">
</form>
</body>
</html>
```

```
Form's action is /handleForm.
This is the URL that will be requested when the user submits the form.
```

```html
<--! Form.html file -->
<html>
<body>

<form action="/handleForm" method="post">

Name: <input name="username">
<p>
I like:<br>
<input type=checkbox name="animal" value="dogs">Dogs <br>
<input type=checkbox name="animal" value="cats">Cats <br>
<input type=checkbox name="animal" value="birds">Birds <br>
<p>
<input type=submit value="Submit form!">
</form>
</body>
</html>
```

The method that we're specifying here is the post method.

```
<--! Form.html file -->
<html>
<body>

<form action="/handleForm" method="post">

Name: <input name="username">
<p>
I like:<br>
<input type=checkbox name="animal" value="dogs">Dogs <br>
<input type=checkbox name="animal" value="cats">Cats <br>
<input type=checkbox name="animal" value="birds">Birds <br>
<p>
<input type=submit value="Submit form!">
</form>
</body>
</html>
```

This would just be the button that the user clicks to submit the form.

The most important part of this example are the action and method of the form.

In this case when we submit the form it'll send an HTTP request for the URL, **handleForm.**

And the method it's going to use is **POST.**

When we use the POST method that means that the data from the form is included in the body of the HTTP request.

So how do we get the form data in our Express app?

```
<--! Form.html file -->
<html>
<body>

<form action="/handleForm" method="post">

Name: <input name="username">
<p>
I like:<br>
<input type=checkbox name="animal" value="dogs">Dogs <br>
<input type=checkbox name="animal" value="cats">Cats <br>
<input type=checkbox name="animal" value="birds">Birds <br>
<p>
<input type=submit value="Submit form!">
</form>
</body>
</html>
```

Reading POST Data in Express

- When a form’s method is “GET”, the data is sent in the URL query parameters

- When a form’s method is “POST”, the data is sent in the **body** of the HTTP request

- To read the body of the HTTP request in Express, use the **body-parser** middleware

- To install it, run: **`npm install body-parser`**

In this example we'll see how we can use the **body-parser** to access the data that was submitted in the form.

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

app.use('/public', express.static('files'));

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

app.use('/handleForm', (req, res) => {
   var name = req.body.username;
   var animals = req.body.animal; // this is an array
    . . .
   res.send('Thank you!');
});
app.listen(3000, () => {
  console.log('Listening on port 3000');
});
```

creating an app using the express function.

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

app.use('/public', express.static('files'));

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

app.use('/handleForm', (req, res) => {
   var name = req.body.username;
   var animals = req.body.animal; // this is an array
    . . .
   res.send('Thank you!');
});
app.listen(3000, () => {
  console.log('Listening on port 3000');
});
```

Because our form was static content, we'll still use express.static middleware to serve static content that starts with the /public URI.

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

app.use('/public', express.static('files'));

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

app.use('/handleForm', (req, res) => {
   var name = req.body.username;
   var animals = req.body.animal; // this is an array
    . . .
   res.send('Thank you!');
});
app.listen(3000, () => {
  console.log('Listening on port 3000');
});
```

We'll create the variable called ('body-parser'), using the keyword require, which is saying we're going to need this body-parser library.

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

app.use('/public', express.static('files'));

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

app.use('/handleForm', (req, res) => {
   var name = req.body.username;
   var animals = req.body.animal; // this is an array
    . . .
   res.send('Thank you!');
});
app.listen(3000, () => {
  console.log('Listening on port 3000');
});
```

body-parser has a middleware function called **urlencoded**, which we'll use to handle forms that are submitted using POST method.

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

app.use('/public', express.static('files'));

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

app.use('/handleForm', (req, res) => {
   var name = req.body.username;
   var animals = req.body.animal; // this is an array
    . . .
    res.send('Thank you!');
});
app.listen(3000, () => {
  console.log('Listening on port 3000');
});
```

the **extended: true** precises that the req.body object will contain values of any type instead of just strings.

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

app.use('/public', express.static('files'));

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

app.use('/handleForm', (req, res) => {
   var name = req.body.username;
   var animals = req.body.animal; // this is an array
    . . .
   res.send('Thank you!');
});
app.listen(3000, () => {
  console.log('Listening on port 3000');
});
```

This will be invoked for every route, which means that when we get to our handleForm request we will have parsed the body of the request.

```
<--! Form.html file -->
<html>
<body>

<form action="/handleForm" method="post">

Name: <input name="username">
<p>
I like:<br>
<input type=checkbox name="animal" value="dogs">Dogs <br>
<input type=checkbox name="animal" value="cats">Cats <br>
<input type=checkbox name="animal" value="birds">Birds <br>
<p>
<input type=submit value="Submit form!">
</form>
</body>
</html>
```

Keep in mind that in our example handleForm was the action of submitting our form.

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

app.use('/public', express.static('files'));

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

app.use('/handleForm', (req, res) => {
   var name = req.body.username;
   var animals = req.body.animal; // this is an array
    . . .
   res.send('Thank you!');
});
app.listen(3000, () => {
  console.log('Listening on port 3000');
});
```

The property would be username, because in our form we called that field username.

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

app.use('/public', express.static('files'));

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

app.use('/handleForm', (req, res) => {
   var name = req.body.username;
   var animals = req.body.animal; // this is an array
    . . .
   res.send('Thank you!');
});
app.listen(3000, () => {
  console.log('Listening on port 3000');
});
```

all of the checkboxes had the same name. So they would be grouped together.
And in general, if we select more than one, this would be an array.

Open this file
http://127.0.0.1:3000/public/form.html

and submit the form.
Check the terminal for output

Serving static files in Express

Read this page
https://expressjs.com/en/starter/static-files.html

So we've seen different ways to get data from the user.

But now how do we send data back based on what the user sent to us?

We're going to see that in the next lecture.

Summary

- HTTP Request **query** properties: key/value pairs that come from URL
- HTTP Request **param** properties: key/value pairs that come from parameterized URL
- HTTP Request **body** properties: input data from form submitting using POST method