

Homework #4

Express with House of the Dragon

Overview

Repository Creation

👁️ Click on this link: <https://classroom.github.com/a/g6JDQCcv> (<https://classroom.github.com/a/g6JDQCcv>) to accept this assignment in GitHub classroom.

- This will create your homework repository
- Clone it once you've created it

Description

Create a site that adds and displays dragons in the show House of the Dragon. Call it: "House of the Dragon". In this homework you'll be working with:

- serving static files
- middleware
- handling forms, both GET and POST
- sessions

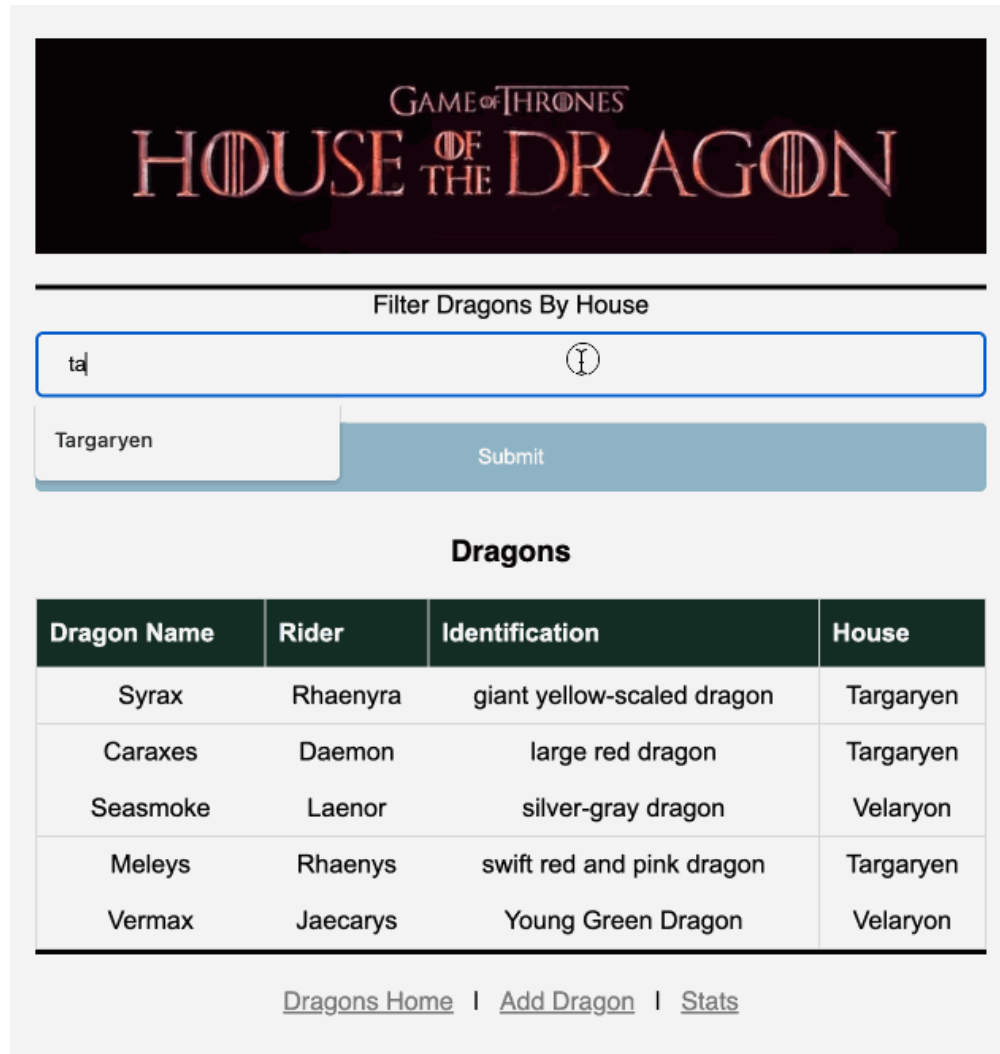
You'll be creating 3 pages:

- **home** - / : displays a list of all of dragons added to the site; can be filtered by the house to which dragons belong.
- **dragon** - /dragon : a page that allows a user to add a new dragon to a house.
- **stats** - /stats : display the number of dragons that the user has added during the current browsing session

Your directory layout should look something like the following **once you're done with the assignment**:

```
├── app.mjs
├── package-lock.json
├── package.json
├── .gitignore
├── public
│   ├── css
│   │   └── styles.css
│   └── img
│       ├── header.jpg
│       ├── img1.jpg
│       ├── img2.jpg
│       ├── img3.jpg
│       └── img4.jpg
└── views
    ├── dragon.hbs
    ├── index.hbs
    ├── stats.hbs
    └── layout.hbs
```

Example Interaction



GAME OF THRONES
HOUSE OF THE DRAGON

Filter Dragons By House

ta ⓘ

Targaryen Submit

Dragons

Dragon Name	Rider	Identification	House
Syrax	Rhaenyra	giant yellow-scaled dragon	Targaryen
Caraxes	Daemon	large red dragon	Targaryen
Seasmoke	Laenor	silver-gray dragon	Velaryon
Meleys	Rhaenys	swift red and pink dragon	Targaryen
Vermax	Jaecarys	Young Green Dragon	Velaryon

[Dragons Home](#) | [Add Dragon](#) | [Stats](#)

Submission Process

You will be given access to a private repository on GitHub. It will contain:

1. some images you can use for testing in the `public/img` folder.
2. you'll have to **create your own** `package.json`, `package-lock.json`, `.gitignore`
 - (these are required and part of grading)
 - use `npm init` to create `package.json` (you can just press enter all the way through)
 - remember to put `node_modules` in your `.gitignore` so that it is not included in your repository
3. if `.eslintrc.cjs` isn't present, you can copy over your linting configuration `.eslintrc.cjs` from previous assignments
 - as usual, you'll have to clean up any eslint warnings or modify eslint config to match your coding style
 - remember to lint only your source code `*.mjs`

Push your code to the homework repository on GitHub. Repositories will close, so make sure you push your changes before the deadline.

Make at Least 4 Commits

- Commit multiple times throughout your development process.

- Make at least 4 separate commits - (for example, one option may be to make one commit per part in the homework).

Part 1 - Setup

Installing Dependencies

- create a `package.json`
- **install** the following **dependencies** (make sure you use the `--save` option), and **no others**:
 - `path`
 - `url`
 - `express`
 - `hbs`
 - `express-session`

`.gitignore`

- create a `.gitignore`
- ignore the following files:
 - `node_modules`
 - any other files that aren't relevant to the project... for example
 - `.DS_Store` if you're on OSX
 - `.swp` if you use vim as your editor
 - etc.

Part 2 - Homepage and Static Files

Enabling Static Files

First, let's make sure you can serve up static content, like css and images.

- you will already have the following directory structure in your project's root directory
 - `public`
 - `public/css`
 - `public/img`
- containing a blank css file in `public/css/styles.css`
- containing images of dragons in `public/img`
- create a basic express application called `app.mjs`; you don't have to define any routes yet...
- just add the appropriate `imports` and `middleware` to enable static file serving:
- test that both the css files and image work
 - for example, try to curl `http://localhost:3000/img/header.jpg`
 - or go that url in your browser

Creating a Home Page

Now that static files are set, create a homepage.

- for the home page, your app should accept `GET` requests on the path, `/`
- set up handlebars
 - get all the requirements and config setup
 - create the appropriate views folder, along with an initial layout file:
 - `views`

- `views/layout.hbs`
- in your `layout.hbs`, drop in the surrounding html that will go on every page
 - pull in your `styles.css` stylesheet
 - include a header containing both your `header.jpg` image and the title of your site, **House of the Dragon**
 - additionally, add a footer that links to all 3 pages in your site:
 - a link to the Dragon home containing list of dragon details (/)
 - a link to **a page to add dragons** (/dragon)
 - a link to a **stats** page (/stats)
 - don't forget `body`, surrounded by triple curly braces!
- in your template for your homepage (you can name this template whatever you want... just make sure you can pull it up later), add the following:
 - an `h3` header that says "Dragons"
- create the appropriate route so that a `GET` request pulls up the rendered template
- add some css to change some styles, (for example change the color of the text, change the font, etc.)

Here's an example of what the page could look like (you don't have to use the same exact styles, but add enough styles so that you can see that the style sheet is integrated correctly):



Filter Dragons By House

Submit

Dragons

Dragon Name	Rider	Identification	House
Syrax	Rhaenyra	giant yellow-scaled dragon	Targaryen
Caraxes	Daemon	large red dragon	Targaryen
Seasmoke	Laenor	silver-gray dragon	Velaryon
Meleys	Rhaenys	swift red and pink dragon	Targaryen

[Dragons Home](#) | [Add Dragon](#) | [Stats](#)

Part 3 - Middleware

Before we start writing our application, let's activate pre-built middleware and write some new middleware to help with debugging. We'll write middleware functions useful for inspecting and logging out request information:

1. Cookie Parser
2. Request Logger
3. Host Checker

It might be helpful to quickly review the slides on middleware ([../slides/09/middleware.html](#)).

The middleware that you write here should be activated before the express static middleware from the previous part. Make sure that the middleware is written in the order presented here, as middleware functions are called sequentially.

Body Parsing Middleware

Use the express' built-in `express.urlencoded({extended: false})` to parse incoming request bodies that are urlencoded (`name=value&name2=value2`).

See the docs (<https://expressjs.com/en/api.html#express.urlencoded>) or in-class demo code (`../code/forms-example.tar.gz`) for example usage.

Cookie Parsing Middleware, Testing Cookies

Without using pre-built middleware for parsing cookies and giving read access to those cookies through a property on the `Request` object, `req`

This middleware:

- checks the incoming request for a `Cookie` header ...
- and parses name value pairs from the value into a property on the 'req' object called `myCookies`

Implementation:

- in `app.mjs` ...
- create a middleware function with three arguments representing the request, response, and next middleware or route to call
- use `req.get` (or `req.header` (<https://expressjs.com/en/api.html#req.get>) to retrieve the `Cookie` header from the request
 - this will give back `undefined` if the header does not exist
- check out our readings to parse the names and values out of the cookie header (MDN on Cookies (<https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies>) and nczonline's article on cookies (<https://www.nczonline.net/blog/2009/05/05/http-cookies-explained/>))
- create a property called `myCookies` on the request object; it should be initialized as an empty object
- add the names and values parsed from the `Cookie` as properties and values on `req.myCookies`
- don't forget to call `next` at the end of the implementation (no arguments need to be passed into the call)
- activate / register the middleware by passing your middleware function as an argument to `app.use`

To test your new middleware:

1. add a route to your application to create cookies:

```
app.get('/test-cookies', (req, res) => {
  res.append('Set-Cookie', 'bestCookie=oatmeal');
  res.append('Set-Cookie', 'bestBreakfast=ramen');
  res.send('test to make cookies');
});
```

2. temporarily add a line in the route above to show the content of `myCookies` :

```
// temporary check for myCookies
console.log('myCookies', req.myCookies);
```

3. (feel free to adjust the values of these cookies to your personal tastes!)
4. go to this route in your browser
5. the output in your terminal will likely be an empty object (as your browser likely does not have cookies for localhost:3000 yet)

6. in your browser, examine the cookies for `localhost:3000` - you should have both set
7. go to the same route again in your browser; this time, the browser (client) *should* send a `Cookie` header with the cookies set from the previous response
8. view your terminal output to verify that the cookie headers are parsed and logged out on the server side

If the middleware works, remove the `console.log`. We'll add logging to the entire application (not just one path) in the next middleware function.

Logging Middleware

Instead of logging out request info for just one path, let's add it for all paths so that we have information if we need to troubleshoot our application (typically logs are sent to a file, but for us, output to the terminal is adequate). Log out the request that you receive, including:

1. the request method
2. the request path
3. the request's query string
4. the request's body
5. the cookie sent in the request's `Cookie` header
 - 1 cookie per line, with a name and value of each cookie
 - if the cookie's name is `connect.sid`, show `[REDACTED]` instead of its actual value

The output may look like this (depending on what path is being requested, what cookies were set, etc.):

```
Request Method : GET
Request Path : /
Request Query : {}
Request Body : {}
Request Cookies :
  foo=bar
  baz=qux
  connect.sid=[REDACTED]
```

To implement:

1. create a middleware function with three arguments: the request object, response object and the next middleware function / route handler to call
2. in the body, `console.log` information according to the specs above
3. don't forget to call `next` (no arguments)
4. register / activate the middleware with `app.use`

Testing your logging

1. try going to a few paths (like `test-cookies` or any of your static files)
2. check the terminal to see that request info has been logged (the query and body will likely be empty objects)

Host Header Checking Middleware

Finally, using what you know about writing middleware from the previous functions, write middleware that will:

1. examine the incoming HTTP Request `Host:` header
2. if it's not present, then immediately send back a `400` response (this specifies `Bad Request`)
3. if the header is present, then proceed normally (go to the next middleware or route handler)

4. both browsers and curl will automatically send this header, but you can use a configuration option for curl to remove that header from the request:
 - `curl -I localhost:3000 -H 'Host:'`
5. alternatively, if you have netcat, you can craft an http request that has no headers
6. in either case, the response should start with something like this:

```
HTTP/1.1 400 Bad Request
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
.
.
.
```

Part 4 - List of Dragons, Filtering based on House

The homepage should have a list of all dragons added to the site. By default, this list will start with the following content:

```
{dragonName:'Syrax', rider: 'Rhaenyra', identification: 'giant yellow-scaled dragon', house: 'Targaryen'},
{dragonName:'Caraxes', rider: 'Daemon', identification: 'large red dragon', house: 'Targaryen'},
{dragonName:'Seasmoke', rider: 'Laenor', identification: 'silver-gray dragon', house: 'Velaryon'},
{dragonName:'Meleys', rider: 'Rhaenys', identification: 'swift red and pink dragon', house: 'Targaryen'}
```

These dragons can be filtered so that your application only shows dragons for a particular house.

Dragons List

Now for some actual content. This page will display the dragons with their details.

Bootstrap the list with some data.

- store all of the dragons data in a global Array of objects...
- each object has four properties:
 - the **dragonName** (just text)
 - the **rider**
 - **identification**
 - **house**
- it should start off as:

Dragons			
Dragon Name	Rider	Identification	House
Syrax	Rhaenyra	giant yellow-scaled dragon	Targaryen
Caraxes	Daemon	large red dragon	Targaryen
Seasmoke	Laenor	silver-gray dragon	Velaryon
Meleys	Rhaenys	swift red and pink dragon	Targaryen

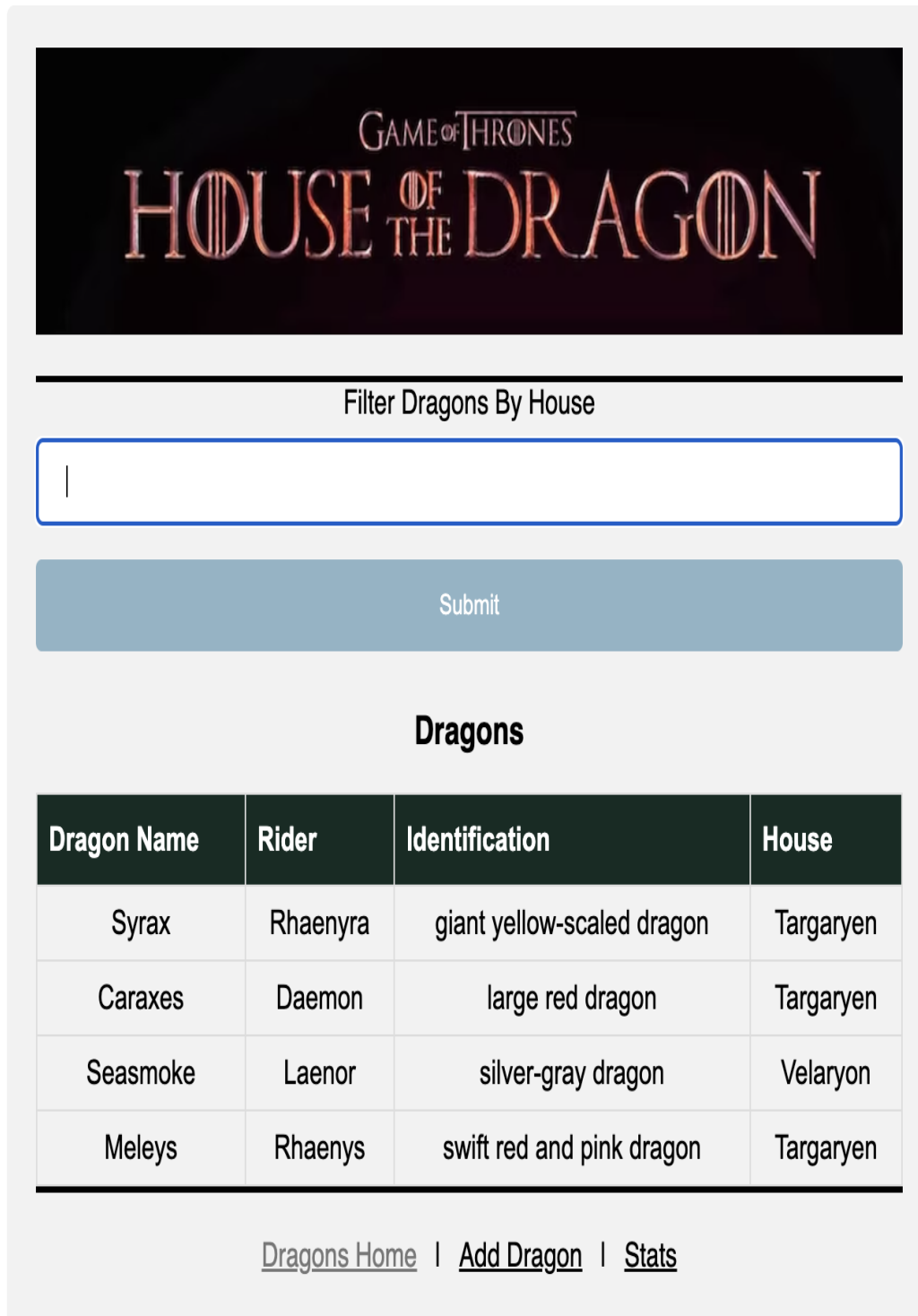
Create the actual page...

- modify your route for your home page (/) so that you render the template with the correct context object (that is, the list of dragon to display)

- in the template, you can iterate through the list of dragon using the `#each` helper
- put each dragon and its details in a table row (`tr`)

Test your page.

- it should look a little like the image below
- again, the styles don't have to match exactly
- (ignore the form for now... you'll set that up next)



GAME OF THRONES
HOUSE OF THE DRAGON

Filter Dragons By House

Submit

Dragons

Dragon Name	Rider	Identification	House
Syrax	Rhaenyra	giant yellow-scaled dragon	Targaryen
Caraxes	Daemon	large red dragon	Targaryen
Seasmoke	Laenor	silver-gray dragon	Velaryon
Meleys	Rhaenys	swift red and pink dragon	Targaryen

[Dragons Home](#) | [Add Dragon](#) | [Stats](#)

Filter by House

Once you have your list of dragons working... add a form that allows you to filter dragon by house. →

- create a form in your `index.hbs` template

- the form should issue a GET request
 - the request should go to the same URL that it's on (still home, /)
 - the form should also have a text field and a submit button
 - remember to give you text field a name!**
- on the server side, modify your route for your home page (/) so that it sends filtered data if the form is submitted
 - how does your route know if the form was submitted?
 - how does the route extract the data from the GET request / form submission?
 - find some way to filter the data
 - send that data to the template
 - if the filter submitted is blank or if there is no filter, display all the dragons.
- here's what the filter interaction should look like:**

GAME OF THRONES
HOUSE OF THE DRAGON

Filter Dragons By House

Submit

Dragons

Dragon Name	Rider	Identification	House
Syrax	Rhaenyra	giant yellow-scaled dragon	Targaryen
Caraxes	Daemon	large red dragon	Targaryen
Seasmoke	Laenor	silver-gray dragon	Velaryon
Meleys	Rhaenys	swift red and pink dragon	Targaryen

[Dragons Home](#) | [Add Dragon](#) | [Stats](#)

The log should look something like this:

GET the home page

```
GET /
=====
Request Method : GET
Request Path : /
Request Query : {}
Request Body : {}
```

GET to submit your filter

```
GET /  
=====  
Request Method : GET  
Request Path : /  
Request Query : {"filter":"Targaryen"}  
Request Body : {}
```

Part 5 - Adding a Dragon

Create a Add Dragon Form

- in **app.mjs** create a new route handler and template for `/dragon`
 - add a form in your template
 - it should have 4 inputs (choose whatever form elements you like) - with appropriate name attributes... you'll see that name in the request body!
 - the dragon name
 - rider
 - identification
 - house
 - ...as well as a submit input
- the form's method should be `POST`
- the action should be empty string `""` or `/dragon` (it's `POST` ing to itself)
- modify **app.mjs** again... by adding a new route so that it accepts `POST` requests on `/dragon`
 - in your callback function for this route...
 - create an object for this new dragon and add it to your global list of dragon.
 - ...after that, redirect to home `/` with a `GET` request
- **here's what the filter interaction should look like:**

GAME OF THRONES
HOUSE OF THE DRAGON

Filter Dragons By House

Submit

Dragons

Dragon Name	Rider	Identification	House
Syrax	Rhaenyra	giant yellow-scaled dragon	Targaryen
Caraxes	Daemon	large red dragon	Targaryen
Seasmoke	Laenor	silver-gray dragon	Velaryon
Meleys	Rhaenys	swift red and pink dragon	Targaryen

[Dragons Home](#) | [Add Dragon](#) | [Stats](#)

The logs should look something like this for the POST, Redirect and GET:

GET the list page.

```
GET /dragon
=====
Request Method : GET
Request Path : /
Request Query : {}
Request Body : {}
```

POST the form (notice the body).

```
POST /dragon
=====
Request Method : POST
Request Path : /dragon
Request Query : {}
Request Body : {"dragonname":"Vermax","rider":"Jacaerys","identification":"Young Green Dragon","house":"Velaryon"}
```

GET the home page (/)...

```
GET /
=====
Request Method : GET
Request Path : /
Request Query : {}
Request Body : {}
```

Part 6 - Stats Page / Using a Session Value

Create a page that shows how many dragons has a user added **during their browsing session**.

Session Setup

First, setup and configure sessions:

- bring in the session module by requiring `express-session`
- set up a some options for your session:

```
var sessionOptions = {  
  secret: 'secrets of targaryen',  
  resave: true,  
  saveUninitialized: true  
};
```

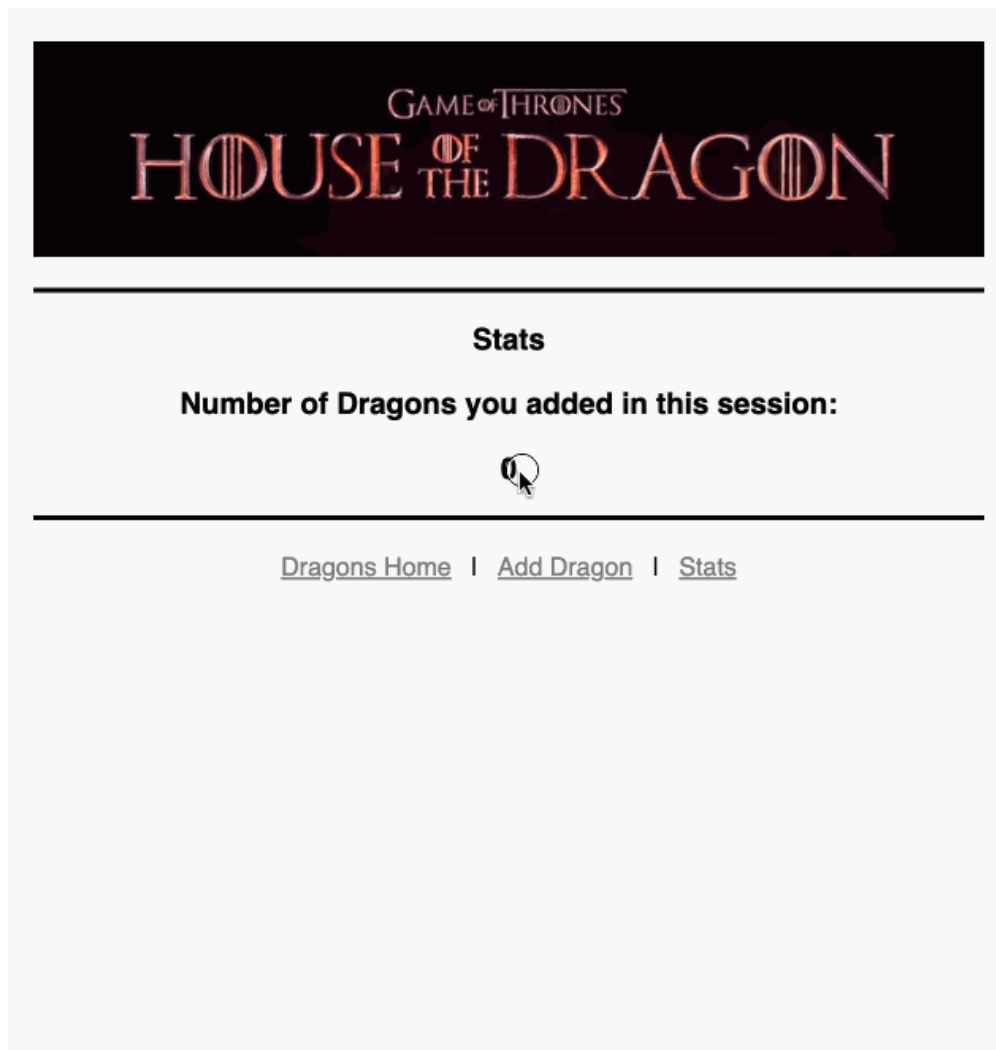
- then use those options for session handling middleware: `app.use(session(sessionOptions));`

Modify your add route (`/dragon`) so that every time a user adds a new dragon, it's counted in their session.

- in the add (`/dragon`) route, count how many dragons a person has added in their session by adding a property to the `req.session` object
- increment the count if there's already a value there
- otherwise, the count should start at 0

The Stats Page

- create a route handler for `/stats`
- create a template for it
- display the session variable that represents the current count of the user's dragons submissions
- the page should look like:

**Test the session management.**

- open your app with one browser... and add some dragons
- the stats page should show the count of dragons added in current session
- open your app in another browser or in private browsing / incognito mode
- check that the count is 0 for this other browser session
- (and of course, make sure that the previous count in your other session was maintained)
- also... regarding sessions/cookies:
 - your server side logging should show there's a session id cookie, but the value should not be shown (on the server)
 - your client, on the other hand, should have `connect.sid` present in the list of cookies