ECS 189
WEB PROGRAMMING

5/10

Useful Piazza comment

From Alex Lujan:

Btw, if any of you get those thrown error messages, just do this, in the terminal:

```
ps aux | grep node
```

you want the 2nd number to the left, next to your kerberos id for the server you previously ran. it's still running, hence the error. you'll get a list like this:

```
kerberosID   2467  0.0  0.5 864534 3456 pts/26   Sl+  18:23   0:00 node server3.js
```

just kill it, using your number via this command:

```
kill -9 2467
```

then you should be able to run your server again

Outage from home

- Someone complained they could not get to the server from home
- This happened to me!
- Turns out using VPN to get to campus library interfered with getting to Digital Ocean; not sure why, but disconnecting VPN fixes it

Good reasons to use VPN

- Can get to O'Reily books for free through library!

PhotoIndex assignments

- Groups of at most 3
- You may do your own design. If you want to, you need to hand in .pdfs showing your proposed design by Fri May 19, and we will grade you on how well your app matches those .pdfs instead of how well they match Dani's
- TAs will just give brief progress grades on Assn5, and a more detailed final grade on finished product in Assn6.

PhotoIndex

- Upload photos to server
- Get Google Cloud Vision API to suggest what is in the images, producing keywords
- Build database of keywords and images
- Let user browse images by keywords
- Let user delete, correct and add keywords

Uploading a file

- Lots of sub-tasks
  - Images are binary files on client
  - Read file into Javascript program
  - They are large, eg. megabytes
  - Send to server in multiple packets
  - The name of the file also needs to be sent

File picking

- This is easy. Use:

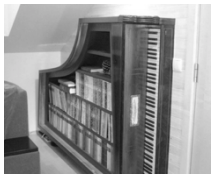
  <input type="file">

- Brings up the familiar file selector dialog box.
- Selected files go into a list that is a property of the DOM object:

```
var selectedFile =
document.getElementById('fileSelector').files[0];
```

File handles

- selectedFile here is a file handle – a data stream (eg what you get when you open a file in say C or Python)
- We could read the file contents from this data stream, then send it
- Instead, we'll use formData object



Forms

- Form is a collection of input items in various formats
- All sent as one single HTTP request when a button is pressed
- FormData object in Javascript lets you package all the data together, as a set of key-value pairs
- We repurpose the formData object to send a file, along with it's name.

Using formData object

```
var formData = new FormData();
formData.append("userfile", selectedFile);
```

- formData contains the key "userfile" with the file handle
- Browser knows to read part of file, sent packet of data, read some more, send more data, … until end of file – this is called serializing
- FormData is (for now) the easiest way to send an image

XMLHttpRequest

- Can be used for POST as well as GET

```
var oReq = new XMLHttpRequest();
oReq.open("POST", url);
oReq.onload = function() {
console.log(oReq.responseText);
  } // equivalent to adding a listener for "load"
  // which occurs when transfer is complete
  oReq.send(formData);
```

Our Server does 3 things

GET request, http://138.68.25.50 :???/pathname

- gives a static file, pathname is based at /public, or 400 message if no such file exist

GET request, http://138.68.25.50.???/query?img=hula

- returns labels associated with image named hula.jpg or hula.jpeg, or 404 message if no such image on server

PUT request to http://138.68.25.50.???/

- uploads an image to /public and responds with a 201 message if successful or 500 message if not

Server

- Has to know how to parse form data and piece together serialized file
- This is possible with Node.js, but even easier with Express.
- Express is a framework (library!) for Node.js
- We'll also use Formidable, an Express framework for handling forms

```
npm install express
npm install formidable
```

A server using express

- Express gives us an alternative syntax for writing a server:

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

var app = express();
```

- app is now an express server object

Static server in express

```
app.use(express.static('public'));
```

- The 4 rather dense lines we had in Node are encapsulated into one nice library function
- "use" in express puts the function inside the parens into a pipeline of functions that are called on the (request, response) pair for a given HTTP request
- Pipeline functions are called "middleware"
- static exits if it finds the file, but not all middleware functions do

Pipeline stage for queries

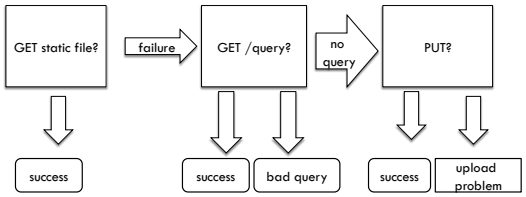
- This is our own query handling bit. Also exits if it succeeds. answer and send404 are our functions.

```
app.get('/query', function (request, response)
  {   console.log("query");
  query = request.url.split("?")[1]; // get query string
  if (query) {
      answer(query, response);   }
  else { send404(request,response);   }});
```

Pipeline stage for PUT requests

```
app.post('/', function (request, response){
  var form = new formidable.IncomingForm();
  form.parse(request);   // does all the work
  // two callback functions
  form.on('fileBegin', function (name, file)
      {   file.path = __dirname + '/public/' +
      file.name;   });   // direct to file in /public
  form.on('end', function ()
      {  console.log('success');
        send201(response,'recieved file'); }); });
```

Pipeline so far



- What is still missing?