Human-Computer Interaction Design

COGS120/CSE170 - "Intro. HCI"

Instructor: Philip Guo

Lab 6 - Connecting frontend and backend without page reloads (2016-11-03)

by Michael Bernstein, Scott Klemmer, and Philip Guo

So far: to fetch new data from the server, your web browser must make a separate URL request and *reload* the entire webpage

Today's goal: to send and receive data from the server without reloading your entire webpage

AJAX
GET and POST

Lab roadmap

Portfolio website

Browser

Server-side

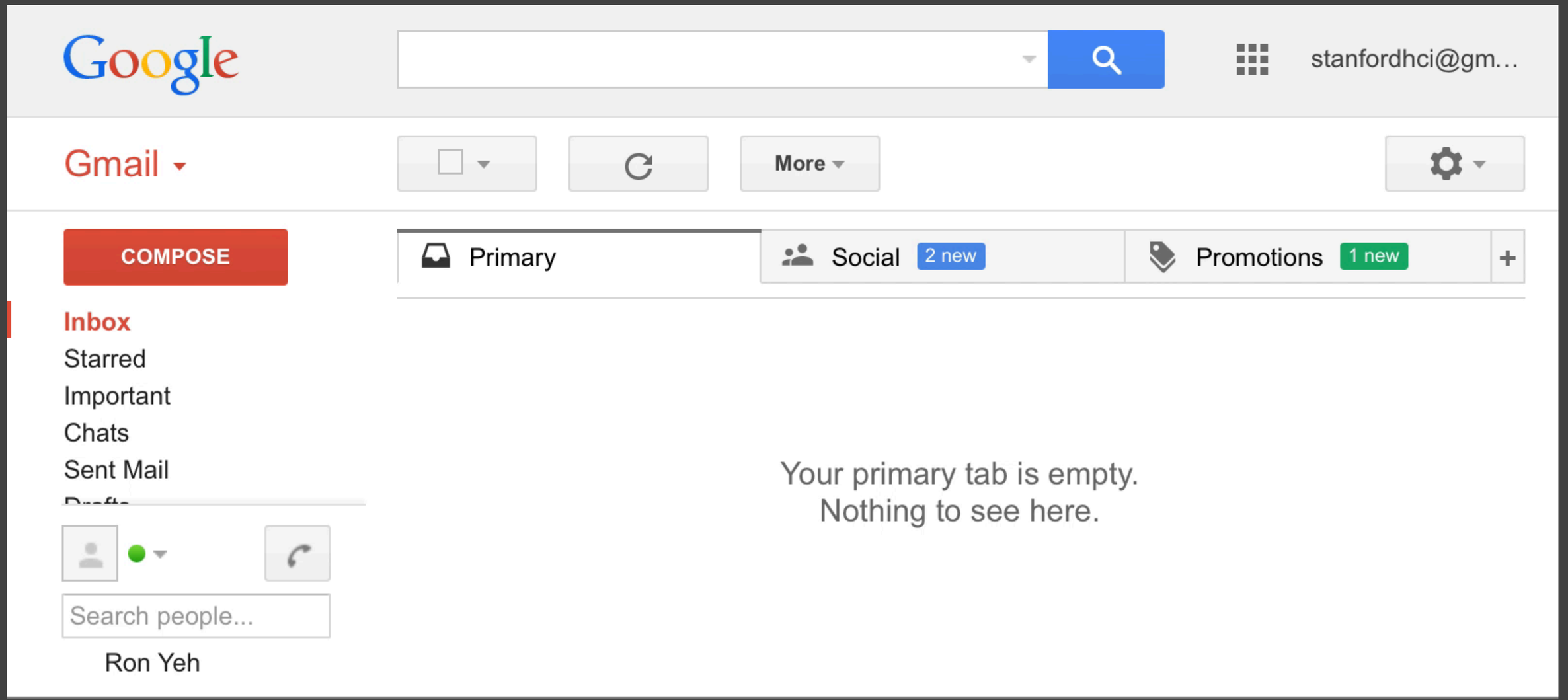
Frontend development

- HTML for page structure: Lab 1
- CSS for styling: Lab 2
- JavaScript for interaction: Lab 3
- AJAX for updates w/out reload: Lab 6

Backend development

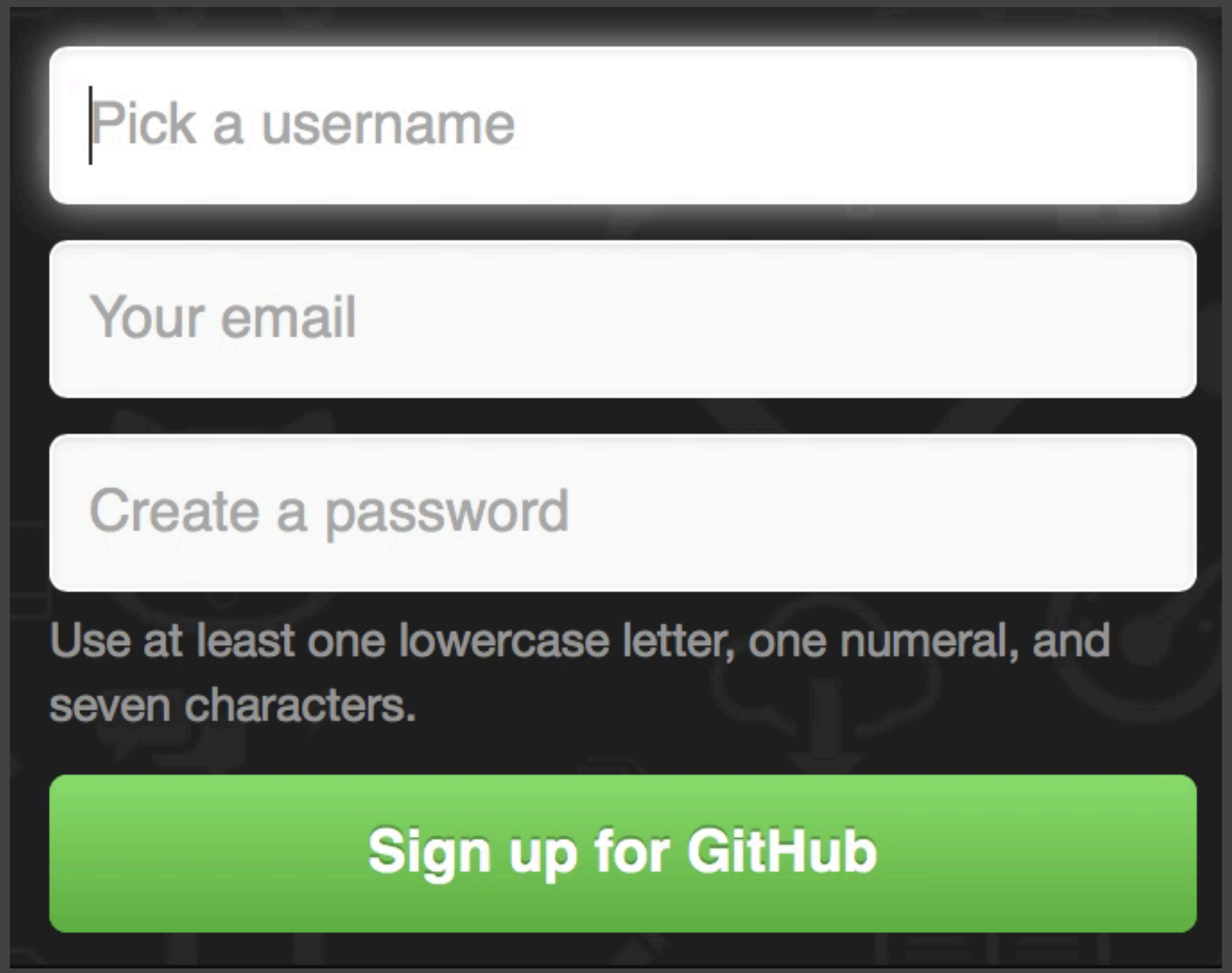
- Express server-side web framework for Node.js: **Lab 4**
- Data storage (future)

Why do this?



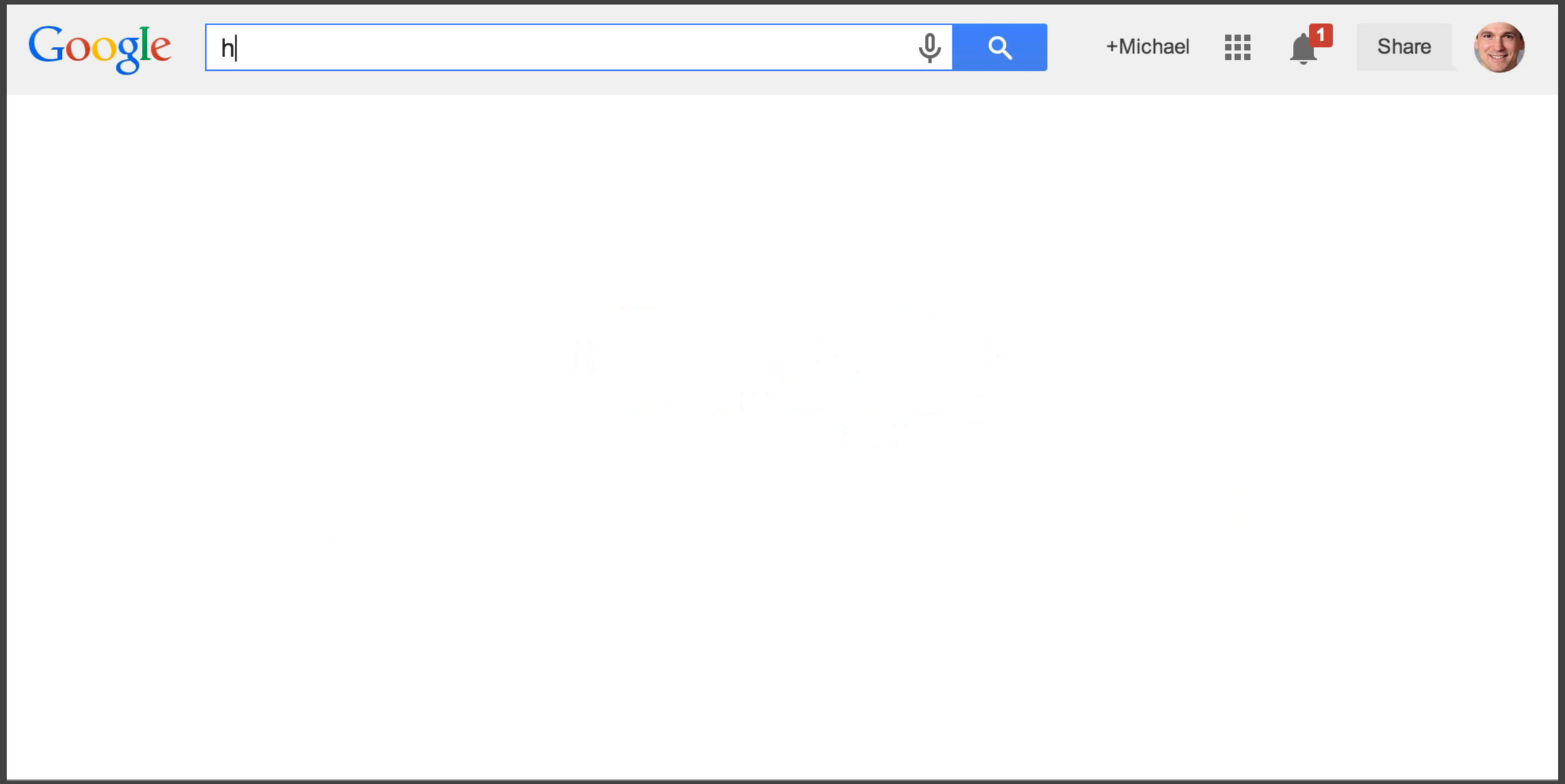
Gmail inbox update

Why do this?



Github username checking

Why do this?



Google instant search

“Hello World” fetch-data-without-reloading example

Get A Project

New Project

First observation: there is no project data embedded in the webpage itself

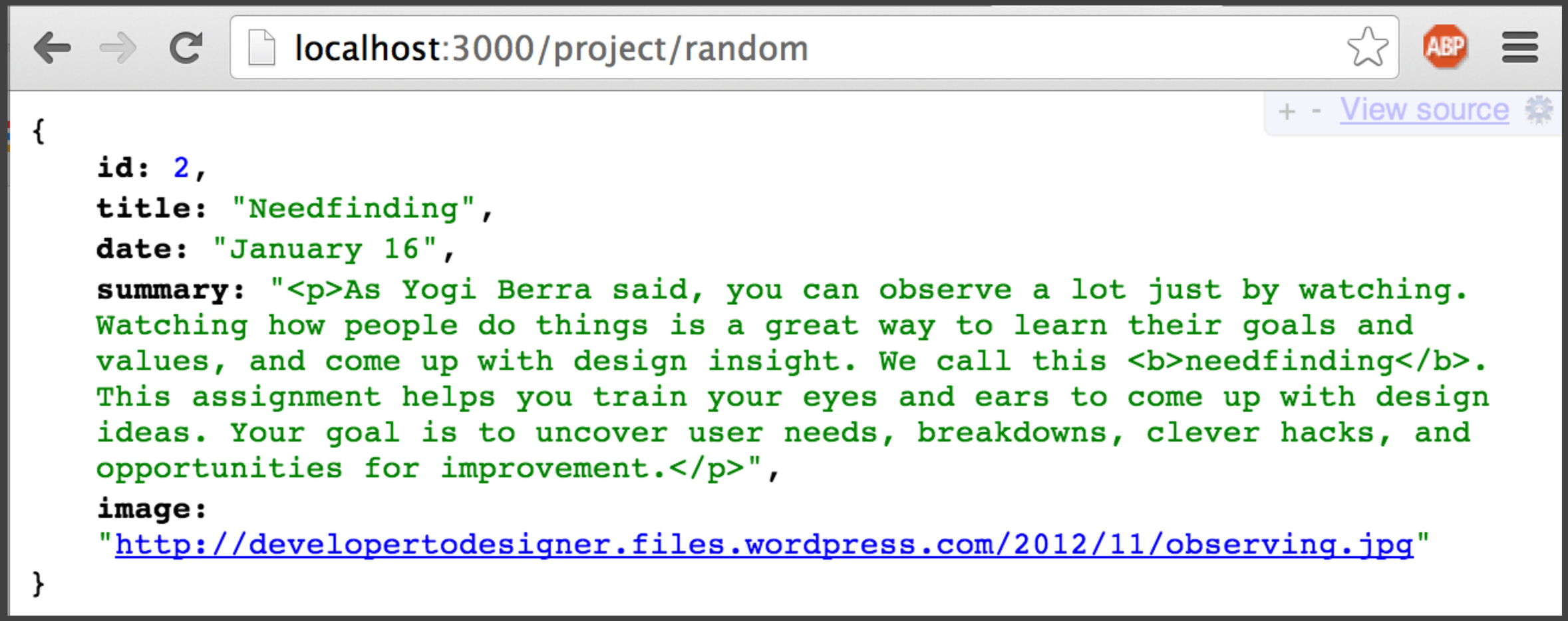
- No JSON object with the project data in introHCI.js
- No hidden `<div>`s in index.html
- No data rendered using index.handlebars

What happens when we click the button?

```
$("#testjs").click(function(e) {
  $.get("/project/random", addProject);
}

function addProject(result) {
  . . .
}
```

What's at the URL in the `$.get()` function?



(keep reloading, and you'll see different random results)

What happens with that JSON data fetched from http://localhost:3000/project/random?

```
$("#testjs").click(function(e) {
  $.get("/project/random", addProject);
}
```

```
function addProject(result) {
  ...
}
```

Callback function:
Just the function name!

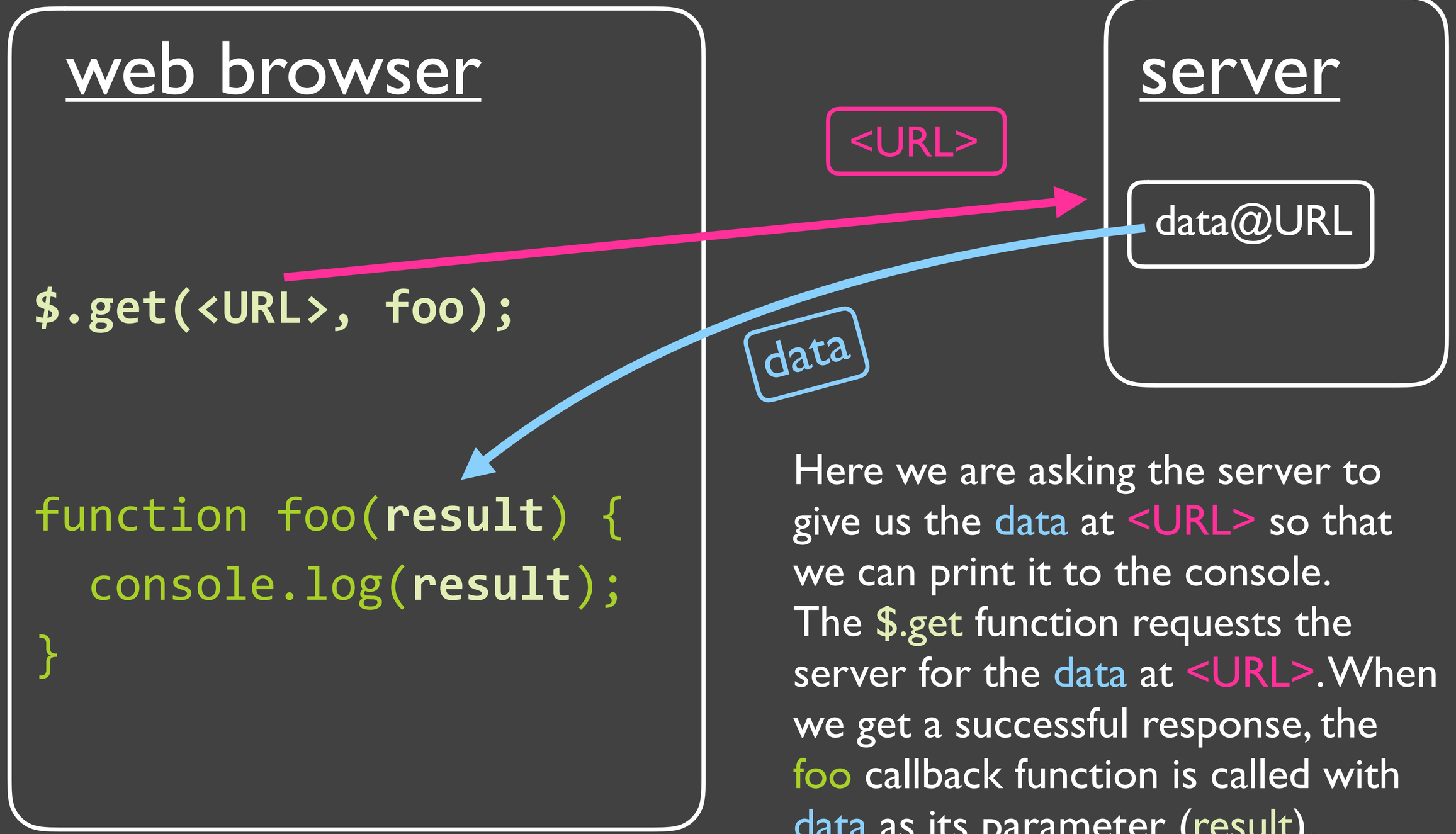
`addProject` not
`addProject(result)`

The callback function uses the JSON result

JSON result properties: `id`, `image`, `title`, `date`, `summary`

```
function addProject(result) {
  console.log(result);
  var projectHTML = '<a href="#" class="thumbnail">' +
    '<img src="' + result['image'] + '" class="img">' +
    '<p>' + result['title'] + '</p>' +
    '<p><small>' + result['date'] + '‘</small></p></a>';

  $("#project-container").html(projectHTML);
  $("#project-description").html(result['summary']);
}
```



Here we are asking the server to give us the data at <URL> so that we can print it to the console. The $.get function requests the server for the data at <URL>. When we get a successful response, the foo callback function is called with data as its parameter (result).

How did the server return that JSON?

see lab6/routes/project.js

```
var projects = require('../projects.json');

exports.projectInfo = function(req, res) {

  var projectID = req.params.id;
  if (projectID == "random") {
    projectID = Math.floor(Math.random() * projects.length) + 1;
  } else {
  projectID = parseInt(projectID);
  }

  var project = projects[projectID-1]; // first project has index 0
  res.json(project); // send JSON back to browser
}
```

concepts

The technique we just introduced is called “AJAX”

Asking the server for data without leaving the current webpage

1. **Javascript** in web browser makes a request

```
$.get(url)
```

```
function {
   ...
   res.json();
}
```

2. Server
returns JSON data
(*not* an HTML webpage)

```
{
   'property': 'value',
   'property2': 'value2'
}
```

3. Javascript
callback function runs
and acts on the data

Why call it AJAX?

Asynchronous

Javascript

And

XML

It used to transfer XML back in the stone age (~2005), but now we use JSON

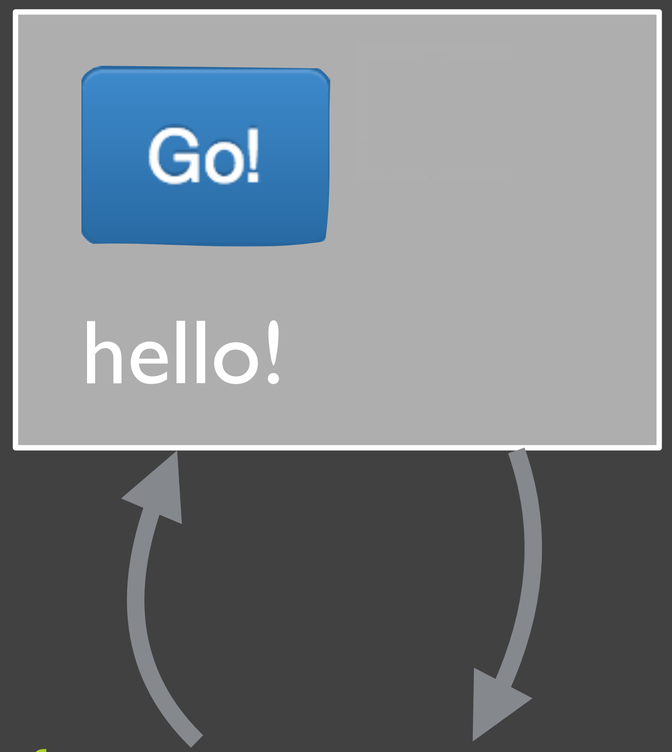
“Asynchronous”?

- The request to the server can take time, so your code keeps running
- Your callback function is called when the request is done
- THIS IS A CORE CONCEPT IN WEB PROGRAMMING, BOTH FRONTEND AND BACKEND. It’s very very very important, and different than most sorts of programming that beginners learn. It can take a while to wrap your head around.

Asynchronous means code keeps running

```
var data;
$('#btn').click(function() {
  $('#wait').show();
  $.get("/data", callbackFn);
  console.log('AJAX');
}

function callbackFn(result) {
  $('#wait').hide();
  data = result['message'] + "!";
  $('body').append(data);
}
```




```
{
  'message': 'hello'
}
```

Console: "AJAX"

Assuming that the system waits: a bug!

```
var data;
$(‘#btn’).click(function() {
  $(‘#wait’).show();
  $.get(“/data”, callbackFn);
  console.log(‘AJAX’);
  $(‘#wait’).hide();
  $(‘body’).append(data);
}

function callbackFn(result) {
  data = result[‘message’] + “!”;
}
```


```
{
  ‘message’: ‘hello!’
}
```

Console: “AJAX”

concepts

Sending data to the server

So far: all data contained in the URL

Example:
`http://localhost:3000/profile/mbernst`

Data (argument) in URL

```
$.get(‘/profile/mbernst’)
```

What if we have a lot of data to send?

Your Message

From: do_not_reply@precisionconference.com

To: msb@cs.stanford.edu

Subject: CSCW 2014 Papers submission 239

Message:

She should have died hereafter;
There would have been a time for such a word.
Tomorrow, and tomorrow, and tomorrow,
Creeps in this petty pace from day to day,
To the last syllable of recorded time;
And all our yesterdays have lighted fools
The way to dusty death. Out, out, brief candle!
Life's but a walking shadow, a poor player
That struts and frets his hour upon the stage
And then is heard no more. It is a tale
Told by an idiot, full of sound and fury
Signifying nothing. — Macbeth (Act 5, Scene 5, lines 17–28)

This much data does not fit inside a URL in most modern browsers (plus, it's super messy even if it fits)

Solution: send JSON using `$.post()`

```
$.post(“/user/sendMessage”,
   {
      “email1": “do_not_reply@precisionconference.com”,
      “email2”: “msb@cs.stanford.edu”,
     “email-content”: “She should have died hereafter;
       There would have been a time for such a
       word. Tomorrow, and tomorrow, and
       tomorrow, Creeps in this petty pace from
       day to day, To the last syllable of
       …"
   },
callbackFunction);
```

`$.get()`

- When you are not impacting the database or application
- e.g., "Get me this data"
- All data goes in URL
- Good because you can bookmark and send URL
- But limited data size due to limited URL length in web browsers

`$.post()`

- When the application takes action based on the data
- e.g., "Post this to Facebook"
- No data in the URL
- Data sent via JSON along with the request
- No size limit on data

Note: these are both part of the jQuery library
(remember the $ dollar sign)

In app.js on the node.js server…

a GET route looks like

```
app.get('someURL/:someId', controller.get_action);
```

a POST route looks like

```
app.post('someURL/:someId', controller.put_action);
```

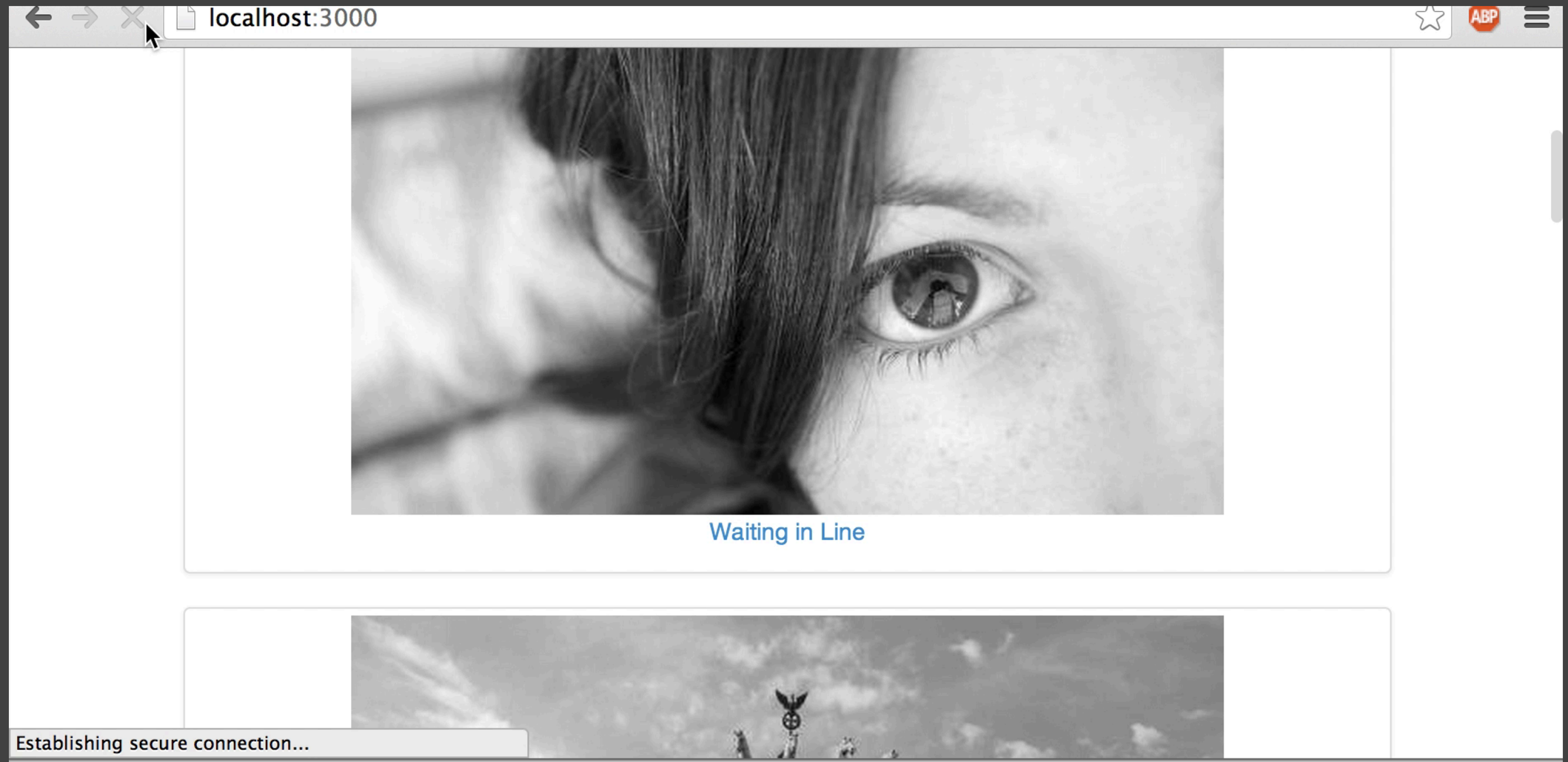
warning: a POST request sent to a GET route URL will be ignored

(Note: these route URLs are sometimes called "***endpoints***")

practice

Your turn: time to make some AJAX requests!

Our goal: dynamically load project info



Get the starter code

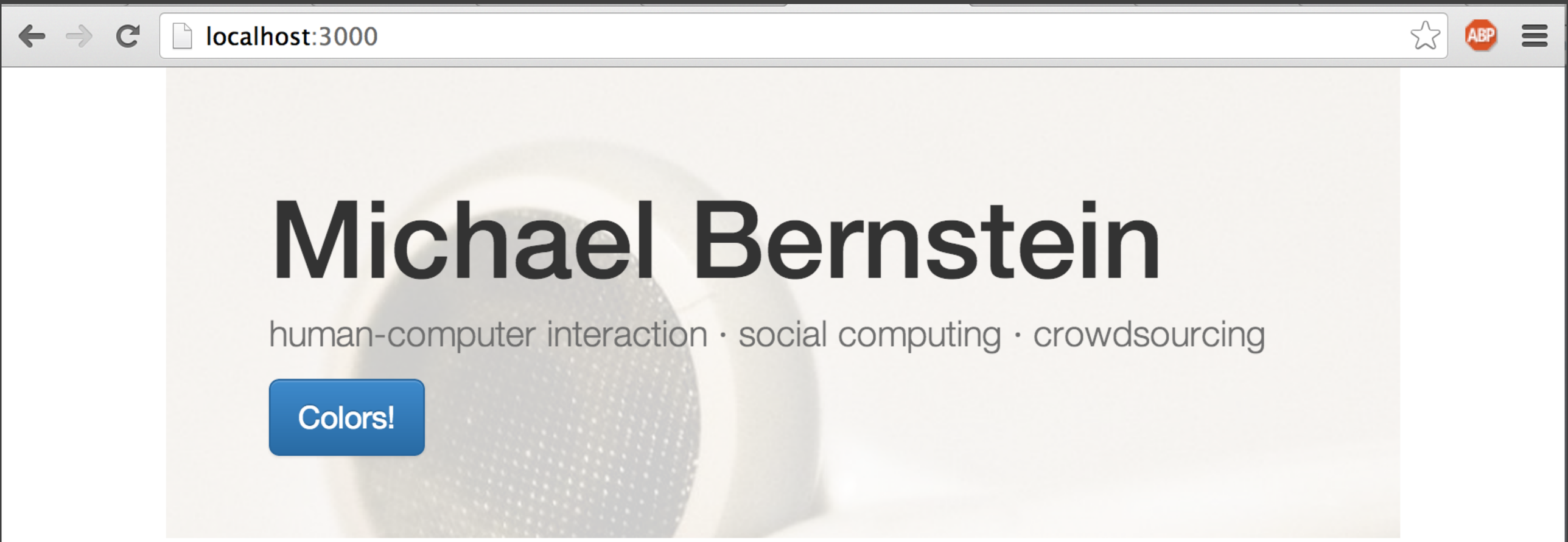
- Fork the repository: https://github.com/pgbovine/lab6
- `git clone` your forked repository into the introHCI directory

Start node.js

- Make sure you're in the lab6 directory, then: `node app.js`

Test node.js: http://localhost:3000

You should see the normal page, as well as Javascript console output when you click on the "Colors!" button or a project.



Open lab6/public/js/introHCI.js

We have added a click listener to the project previews

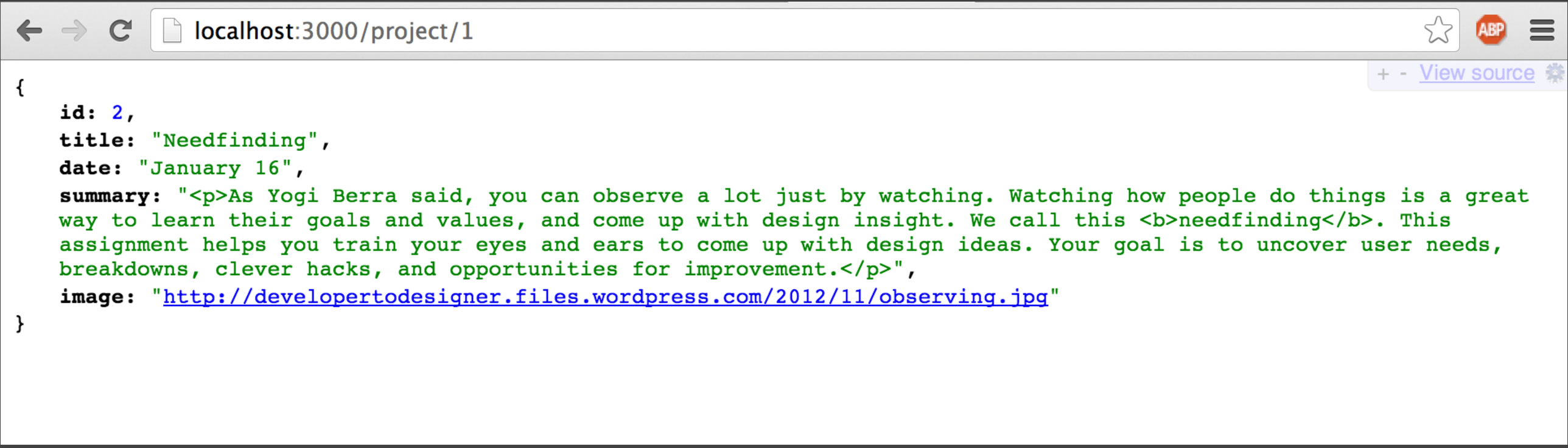
```
function addProjectDetails(e) {
    // Prevent following the link
    e.preventDefault();

    // Get the div ID, e.g., "project3"
    var projectID = $(this).closest('.project').attr('id');
    // get rid of 'project' from the front of the id 'project3'
    var idNumber = projectID.substr('project'.length);

    console.log("User clicked on project " + idNumber);
}
```

Test the project detail endpoint URL in your browser

http://localhost:3000/project/1
http://localhost:3000/project/2
http://localhost:3000/project/3



(Note: route URLs in app.js are sometimes called "*endpoints*")

Call the project endpoint URL via AJAX

- *What do I do?*
  - When the user clicks on a particular project, call the endpoint URL from the previous slide using `$.get`
  - Make sure you customize the URL based on the `idNumber` variable
  - Use `console.log()` to print out the URL that you are calling
  - Create an empty callback function for now; we'll get there soon. It should have an argument (e.g., `result`) for the server response
- *Is it working?*
  - Click on a project, then check the Javascript terminal to find the URL you printed out. It should say "/project/1", "/project/2", etc.
- Error: *"Uncaught ReferenceError: [functionName] is not defined"*
  - Your callback function either doesn't exist, or the name you gave to the `get` function doesn't match the actual name of the function

Write the callback function

- *What do I do?*
  - `console.log` the response data to make sure it is correct
- *Is it working?*
  - Click on a project and see if it outputs the correct data to the console
- *Common errors*
  - *The data is for the wrong project.* Are you sure you are sending the correct project ID number in the URL?
  - *I don't know where the data is coming from.* Did you include a `result` parameter to your callback function for the AJAX result?

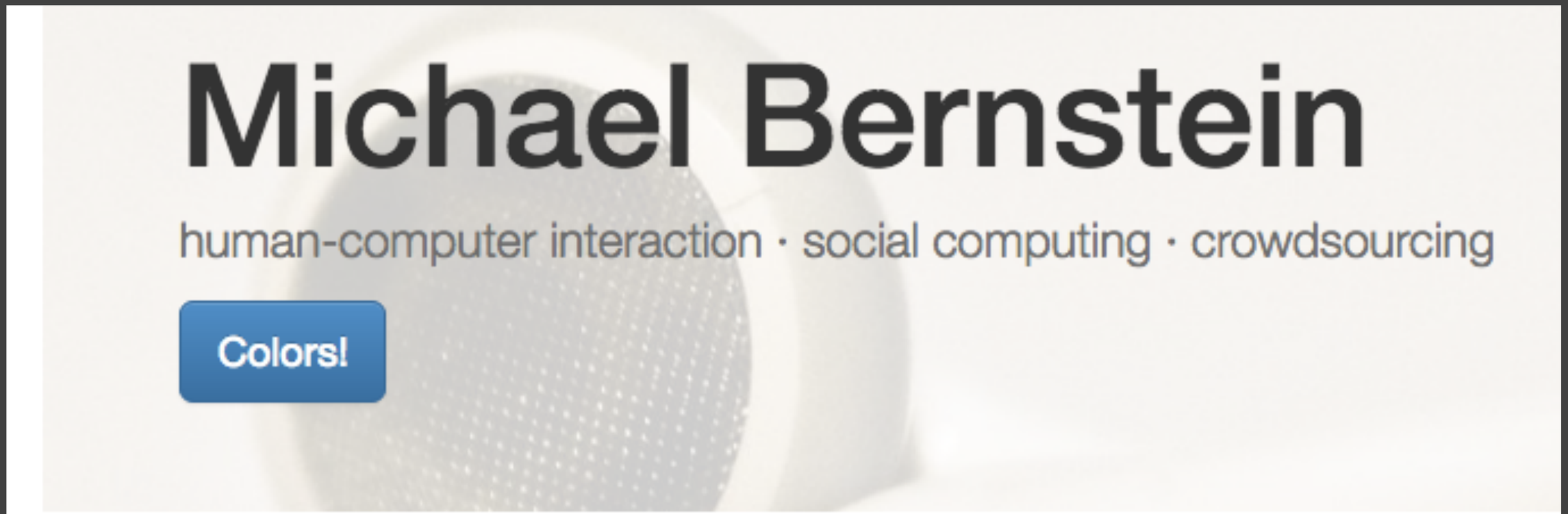
Insert the project details into the page (1)

- *What do I do?*
  - Use jQuery to select the div where we'll be inserting the content. It has class `.details`, but make sure you're only selecting the one that's a descendent of the project div with the right ID. (Really think through what kind of jQuery selector syntax will work here.)
  - Use jQuery's `html()` to change the contents of the `details` div.
  - Test it by inserting debug text like "foo"
- *How do I know if it's working?*
  - Click on the project, and it should insert your debug text "foo"
- *Nothing is happening when I click.*
  - Use the JS console to make sure you're selecting the right div.

Insert the project details into the page (2)

- *What do I do?*
  - Create the HTML string to insert and mix in the results from the AJAX call. You'll want to include the image, a small header with the date, and the summary.
  - Give the image the class `detailsImage` (we wrote a CSS rule)
  - Replace your debug text with the HTML string
- *How do I know if it's working?*
  - Click on the project, and it should insert your project details
- *Nothing is happening when I click.*
  - Look at the Javascript console for errors, and try printing out your HTML string that you're trying to insert.

Now let's make something cool happen when the user clicks the "Colors!" button



Make the AJAX request

- *What do I do?*
  - We will be placing our AJAX endpoint URL at /palette
  - When the user clicks on the "Colors!" button at the top of the page, issue an AJAX GET request to the /palette URL.
  - The code should be very similar to what you wrote for the last part of the lab.
- *How do I know if it's working?*
  - When you click on the "Colors!" button, the Javascript console in Chrome tells you that it can't find the localhost:3000/palette URL.

Prepare the /palette endpoint

- *What do I do?* Make sure you have three pieces of code to create an endpoint at /palette:
  1. Controller file palette.js — we have given you this
  2. Import the controller file near the top of app.js
  3. Define the route/URL at the bottom of app.js
- *How do I know if it's working?*
  - Restart node.js and try to go to localhost:3000/palette. Make sure it displays our debug string.

Return a color palette from the endpoint

- *What do I do?*
  - Edit palette.js to return JSON for the chosen palette. To do this, first get rid of the `res.send()` line. Then, use `res.json()` and pass it the JSON object it should return.
  - Return the one randomly-chosen palette the code has selected.
- *How do I know if it's working?*
  - Go to localhost:3000/palette and you should see JSON data about a randomly-chosen color palette

Hook up your callback

- *What do I do?*
  - Make the AJAX call whenever the user clicks the "Colors!" button
  - In the callback function, find the hex colors within the javascript object that you receive from server. Put that array into `var colors`.
  - Use our provided Javascript below to apply those colors to the page using jQuery's `css` function … copy and paste:

```
$('body').css('background-color', colors[0]);
$('.thumbnail').css('background-color', colors[1]);
$('h1, h2, h3, h4, h5, h5').css('color', colors[2]);
$('p').css('color', colors[3]);
$('.project img').css('opacity', .75);
```

- Tip: think hard about where each piece of code should go. Remember that AJAX is "asynchronous," so which code runs in which order may be counter-intuitive.

All set! Now try out your color disco by clicking on the “Colors!” button repeatedly.

Michael Bernstein

human-computer interaction · social computing · crowdsourcing

Colors!

Projects