

Assignment 5: Single-Page Applications

CS390P: Web Application Architecture

John Samson

December 2nd, 2018

Abstract

So far, we've used the Rails framework to quickly create a running web application, in which we used multiple pages to display registration information for a university. In order to create a more seamless experience for the user, today we will update our web application to be a *single-page application (SPA)*. Previously we used Rails layouts and scaffolds to create static HTML pages. Here we will have *JavaScript* parse Rails' built-in *JavaScript Object Notation (.json)* API to dynamically create the HTML for the different subpages of our site at runtime. This paper will serve as an introduction to single-page applications, as well as JavaScript, the JQuery library, and JavaScript Object Notation (.json).

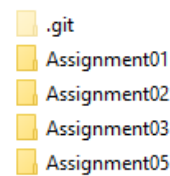
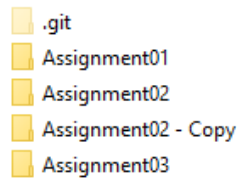
Introduction

Our goal will be to update our running web application from Assignment 2 to be a single-page application using JavaScript and the JQuery library to dynamically create HTML on the client side at runtime. There are dozens of different libraries and frameworks for creating single-page applications. We chose JQuery due to its popularity and relative ease of use. We will begin with a very simple template file provided by Dr. Steve Beaty, available here: <https://github.com/drsjb80/SPAs/blob/master/jqspa.htmlb> (Beaty, 2018). We will create simple JavaScript functions to parse Rails' built-in JSON API and build our HTML "from scratch". This paper is merely an introduction to JavaScript and will leave implementation of our Search bar and database Create/Update/Display (CUD) operations as an advanced exercise for the reader.

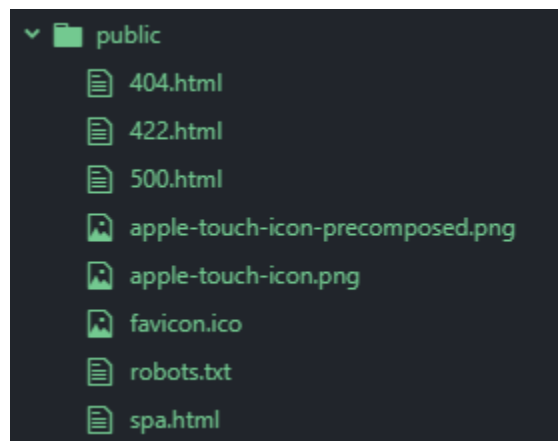
A Simple SPA

Before we dive head first into the world of JavaScript and jQuery, let's setup our work environment and template SPA, as well as add our new project to revision control.

- First let's make a backup of our work from Assignment 2. Copy the 'Assignment 2' folder from your local repository and rename it to 'Assignment 5'.



- Let's add our template jQuery SPA (provided by Dr. Steve Beaty, available here: <https://github.com/drsjb80/SPAs/blob/master/jqspa.html>) to our Rails project. Navigate to the '/public' folder within your Rails application and create a new text document and save it as 'spa.html'. Either copy or clone 'jqspa.html' from Dr. Beaty's repository and save it to 'spa.html'. Note that we don't have to do much involving Rails here, we're simply creating a SPA with our existing model-view-control architecture.



- Now let's add our new Project, including template SPA to our own repository:

```
C:\CS390P-Web-Application-Architecture>git add .
```

```
C:\CS390P-Web-Application-Architecture>git commit -am "Assignment 5 initial commit"
```

```
C:\CS390P-Web-Application-Architecture>git push origin master
```

- Let's take a quick look at the template file:

```
<button onclick="fetch('students', ['name', 'advisor_id'])">Students</button>  
<button onclick="fetch('advisors', ['name'])">Advisors</button>
```

- Our application doesn't have 'advisors', so let's change that to 'courses'. We also don't have 'advisor_id', so let's remove that too. Update the 'button' tags to the following.

```
<button onclick="fetch('students', ['name'])">Students</button>
<button onclick="fetch('courses', ['name'])">Courses</button>
```

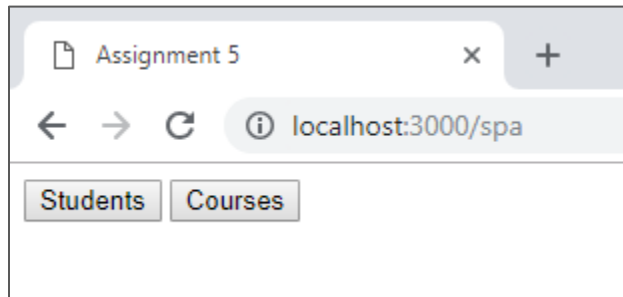
- Feel free to change the title of your SPA:

```
<title>Assignment 5</title>
```

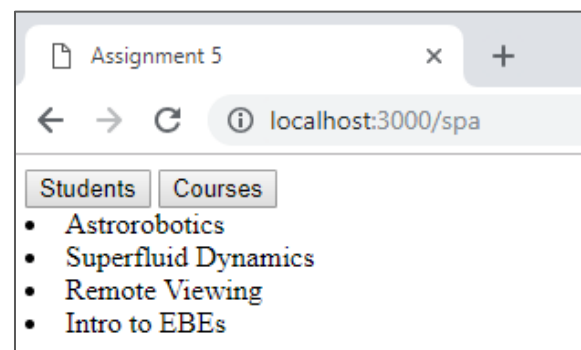
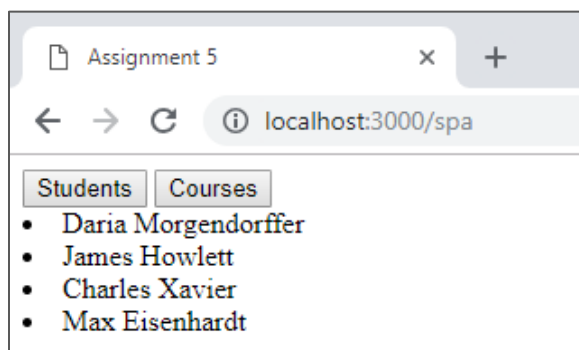
- Now run the rails server to view the template SPA.

```
C:\CS390P-Web-Application-Architecture\Assignment05>rails s
```

- We can see we have two buttons that rendered properly:



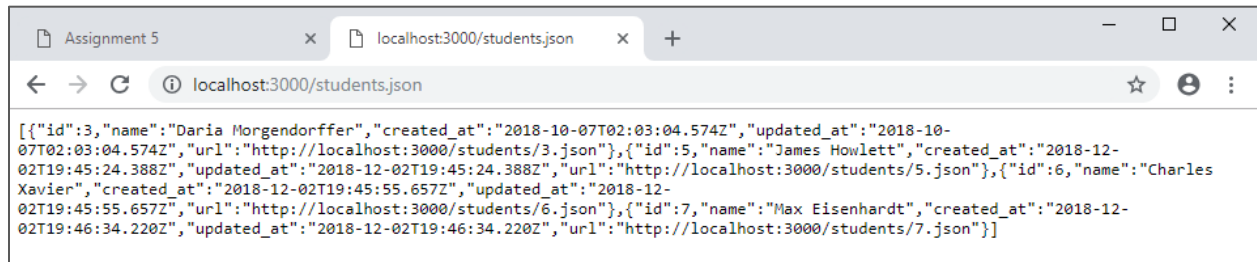
- Clicking on either of them shows Students and Courses as we would expect:



While you might guess that our SPA is querying the SQLite3 database, you would be incorrect in assuming so. To understand what's happening here, let's take a look at our JavaScript and JSON API built into Rails.

- Navigate to <http://localhost:3000/students.json> to view each Student object in JavaScript Object Notation (JSON).

- We can see that 'students.json' has two fields, "id" and "name":



```
[{"id":3,"name":"Daria Morgendorffer","created_at":"2018-10-07T02:03:04.574Z","updated_at":"2018-10-07T02:03:04.574Z","url":"http://localhost:3000/students/3.json"}, {"id":5,"name":"James Howlett","created_at":"2018-12-02T19:45:24.388Z","updated_at":"2018-12-02T19:45:24.388Z","url":"http://localhost:3000/students/5.json"}, {"id":6,"name":"Charles Xavier","created_at":"2018-12-02T19:45:55.657Z","updated_at":"2018-12-02T19:45:55.657Z","url":"http://localhost:3000/students/6.json"}, {"id":7,"name":"Max Eisenhardt","created_at":"2018-12-02T19:46:34.220Z","updated_at":"2018-12-02T19:46:34.220Z","url":"http://localhost:3000/students/7.json"}]
```

- Courses expectedly has more fields:

```
[{"id":6,"name":"Astrorobotics","department":"EGR","number":5600,"credit_hours":9,"created_at":"2018-10-14T00:07:40.264Z","updated_at":"2018-10-14T00:07:40.264Z","url":"http://localhost:3000/courses/6.json"}, {"id":7,"name":"Superfluid Dynamics","department":"PHY","number":4500,"credit_hours":7,"created_at":"2018-10-14T00:08:05.743Z","updated_at":"2018-10-14T00:08:05.743Z","url":"http://localhost:3000/courses/7.json"}, {"id":8,"name":"Remote Viewing","department":"PSY","number":2000,"credit_hours":3,"created_at":"2018-12-02T19:50:01.869Z","updated_at":"2018-12-02T19:50:01.869Z","url":"http://localhost:3000/courses/8.json"}, {"id":9,"name":"Intro to EBES","department":"EXO","number":2400,"credit_hours":3,"created_at":"2018-12-02T19:50:51.012Z","updated_at":"2018-12-02T19:50:51.012Z","url":"http://localhost:3000/courses/9.json"}]
```

- We can see our one-to-many relationship here, Sections references 'course_id':

```
[{"id":14,"course_id":6,"semester":"Spring 2080","number":1,"room_number":260,"created_at":"2018-10-14T00:35:18.400Z","updated_at":"2018-10-14T00:35:18.400Z","url":"http://localhost:3000/sections/14.json"}, {"id":15,"course_id":7,"semester":"Fall 2079","number":1,"room_number":300,"created_at":"2018-10-14T00:35:29.565Z","updated_at":"2018-10-14T00:35:29.565Z","url":"http://localhost:3000/sections/15.json"}, {"id":18,"course_id":8,"semester":"Fall 2079","number":1,"room_number":149,"created_at":"2018-12-02T20:05:06.806Z","updated_at":"2018-12-02T20:05:06.806Z","url":"http://localhost:3000/sections/18.json"}, {"id":19,"course_id":9,"semester":"Spring 2080","number":1,"room_number":295,"created_at":"2018-12-02T20:05:26.896Z","updated_at":"2018-12-02T20:05:26.896Z","url":"http://localhost:3000/sections/19.json"}]
```

- Likewise, Enrollments references both 'section_id' and 'student_id':

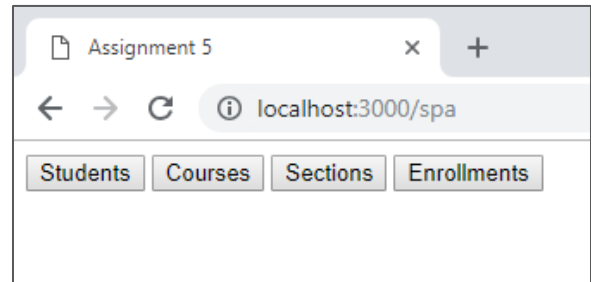
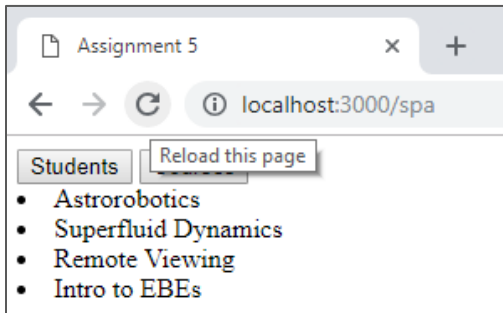
```
[{"id":13,"section_id":18,"student_id":5,"created_at":"2018-12-02T20:06:49.088Z","updated_at":"2018-12-02T20:06:49.088Z","url":"http://localhost:3000/enrollments/13.json"}, {"id":14,"section_id":18,"student_id":3,"created_at":"2018-12-02T20:06:55.977Z","updated_at":"2018-12-02T20:06:55.977Z","url":"http://localhost:3000/enrollments/14.json"}, {"id":15,"section_id":14,"student_id":6,"created_at":"2018-12-02T20:07:03.873Z","updated_at":"2018-12-02T20:07:03.873Z","url":"http://localhost:3000/enrollments/15.json"}, {"id":16,"section_id":14,"student_id":7,"created_at":"2018-12-02T20:07:12.382Z","updated_at":"2018-12-02T20:07:12.382Z","url":"http://localhost:3000/enrollments/16.json"}, {"id":17,"section_id":19,"student_id":5,"created_at":"2018-12-02T20:07:25.945Z","updated_at":"2018-12-02T20:07:25.945Z","url":"http://localhost:3000/enrollments/17.json"}]
```

- Now would be a good time to update our Buttons to create a simple Navigation Bar. Let's go ahead 'fetch' any possibly useful information in case we might need it later. Add the following to your 'spa.html' file:

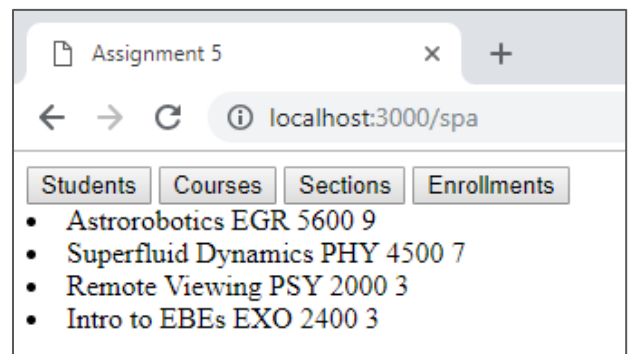
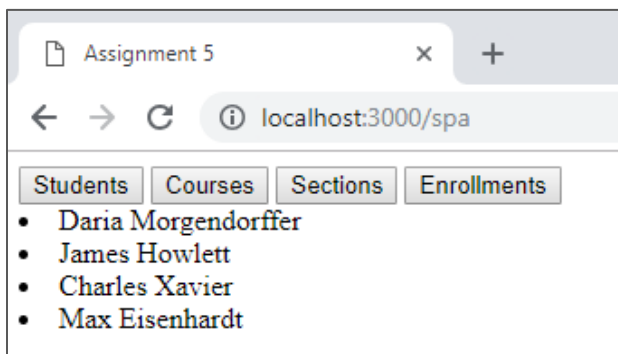
```
<button onclick="fetch('students', ['name'])">Students</button>
<button onclick="fetch('courses', ['name','department','number','credit_hours'])">Courses</button>
<button onclick="fetch('sections', ['course_id','semester','number','room_number'])">Sections</button>
<button onclick="fetch('enrollments', ['section_id','student_id'])">Enrollments</button>
```

- Sections and Enrollments don't have a 'name' field in their respective JSON objects, so let's not bother fetching them. They instead contain the *references* to the names of other objects. We'll soon create a JavaScript function to navigate to the referenced page and find the necessary information by ID.

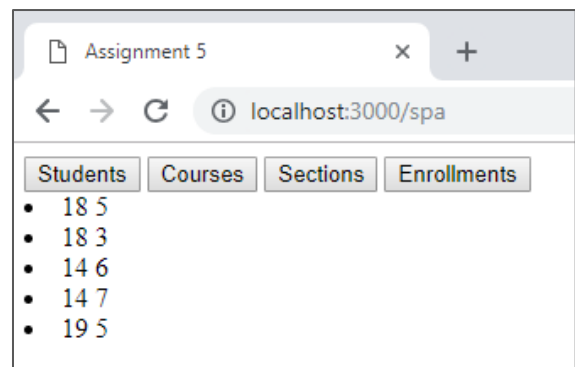
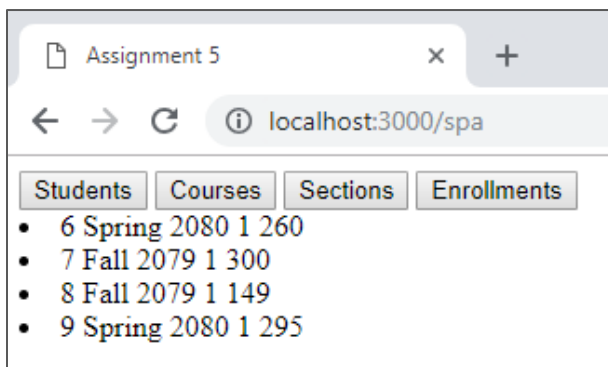
- Reload the page to see the changes we just made. Note that we do not need to restart the Rails server.



- Students has not changed, though Courses shows some new information:



- Sections and Enrollments both need a 'name'. Due to the way we created our one-to-many relationship in our model, JSON stores only the ID of the referenced object. Next we must write a JavaScript function to find and retrieve the relevant Course, Student, and Section names.



We've successfully accessed our web applications JSON API to retrieve information from an HTML table in JSON format. We'll continue our work with a closer look at our JavaScript functions, and learn more about how we parse JSON objects to dynamically build our sub-pages at runtime.

Adding a Header

Before we begin fixing our referenced data table items, let's begin by recreating something far simpler. Let's see if we can add some functionality to our script to display the relevant HTML header for each subpage. Open your 'spa.html' file and take a closer look at our JavaScript 'fetch' function.

- When we click a button on our SPA page, we specified the JavaScript *onclick* to call the following 'fetch' function. Note that we iterate through each JSON object and then through each *field* within that object:

```
<script>
var fetch = function(which, fields) {
  $("#list").empty();
  $.get("http://localhost:3000/" + which + ".json", true)
  .done(function(json) {
    for (item in json) {
      lis = "<li>";
      for (field in fields) {
        lis += json[item][fields[field]] + " ";
      }
      lis += "</li>";
      $("#list").append(lis);
    }
  })
  .fail(function(json){ console.log("fail"); })
};
</script>
```

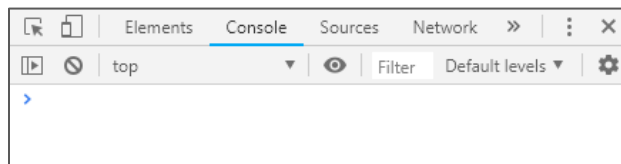
- A powerful and essential tool for web developers is the *web console*. Let's add a simple statement to print the value of the *which* parameter used in our fetch function, to see *which* sub-page we're currently accessing:

```
.done(function(json) {
  console.log("which = ", which);
```

- Right click anywhere in your browser with your SPA running, then select 'Inspect' to access developer tools.
- Note that while we are using Chrome here, the following steps should be quite similar for other browsers as well.

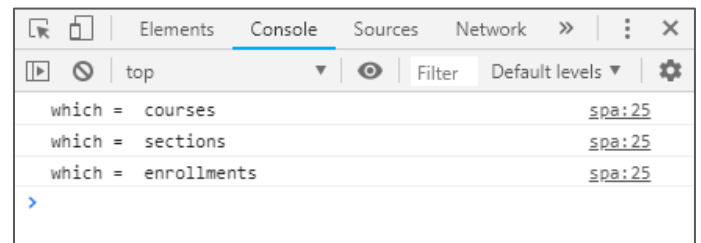
Back	Alt+Left Arrow
Forward	Alt+Right Arrow
Reload	Ctrl+R
Save as...	Ctrl+S
Print...	Ctrl+P
Cast...	
Translate to English	
View page source	Ctrl+U
Inspect	Ctrl+Shift+I

- Click 'Console' to access the web console.

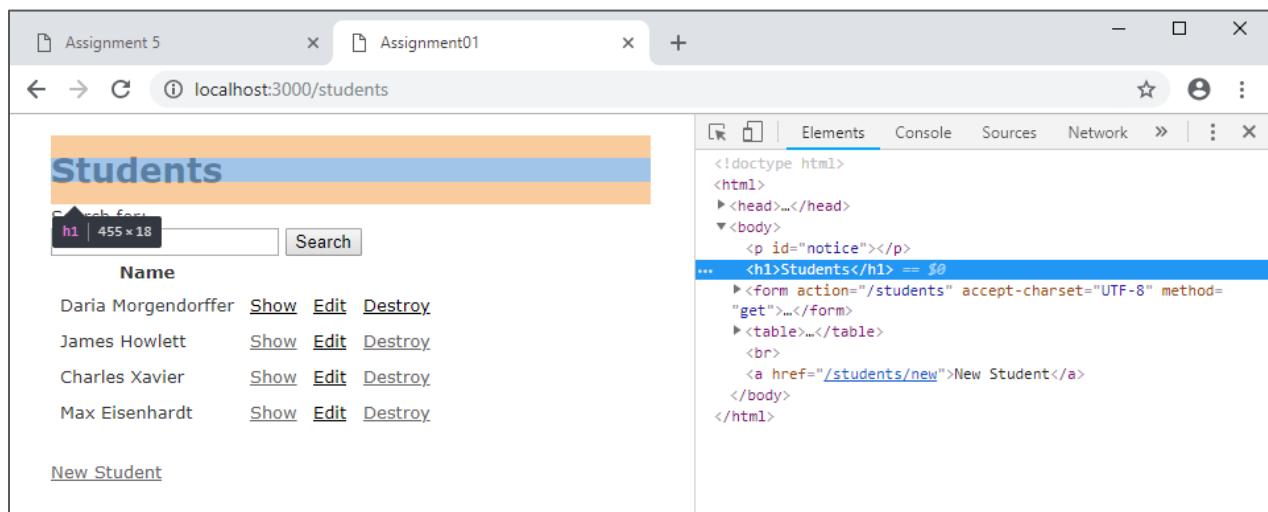


- Navigating through the different sub-pages

executes our *console.log* statement:



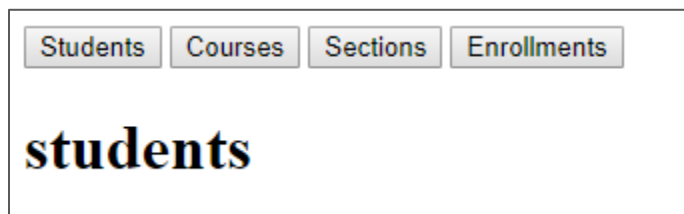
- Now let's recreate the HTML header within our SPA. Navigate to any '/index' page in our *old* application and 'Inspect' the HTML header. We can see that the 'Students' header here is wrapped within a `<h1>` tag.



- Let's do our best to recreate it. Add the following line before the `'for (item in json)'` loop in your JavaScript. Note that we'll store our parsed JSON objects and added HTML tags as a jQuery *list* (`$("#list")`) object, which we can *append* strings to. Here we use the *which* parameter for our header:

```
$("#list").append("<h1>" + which + "</h1>");
```

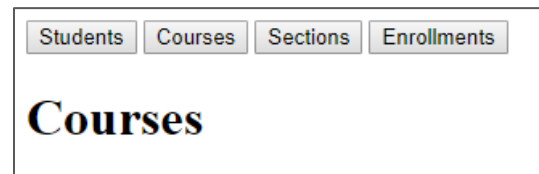
- This looks to be the appropriate header style, though lower case isn't quite formal enough for a header.



- Change the preceding line to the following to capitalize the header string. We create a temporary variable ‘tmp’ simply because we use the *slice* method to capitalize our string. We don’t want to destroy *which*, so let’s *slice* ‘tmp’ instead:

```
tmp = which;
$("#list").append("<h1>" + tmp.charAt(0).toUpperCase() + tmp.slice(1) + "</h1>");
```

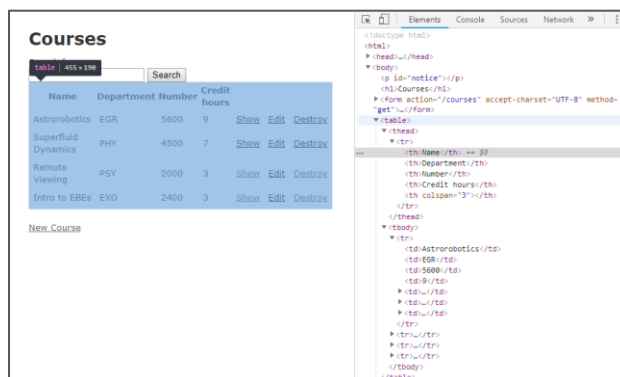
- Each sub-page now has an appropriate HTML Header:



Recreating a Table

So far, we’ve created a simple navigation bar using Buttons, as well as a *parameterized* sub-page Header. Let us take on more adventurous work and begin recreating our data table for each sub-page. Right now our simple template SPA presents information as an unordered list, which isn’t really a good fit for our data. We’ll need to create and populate an HTML table including table headers, and accommodate for the references between data tables we’ve created.

- Navigate to any page in our old Assignment 2 project, then ‘Inspect’ to see how our HTML table is formatted:



- Here we can see our table wrapped in a `<table>` tag, with `<thead>` and `<tbody>` for table head and body, respectively. Individual table heading items are specified as `<th>` table header items, and individual cells are specified as `<td>` table data items.

Table Headers

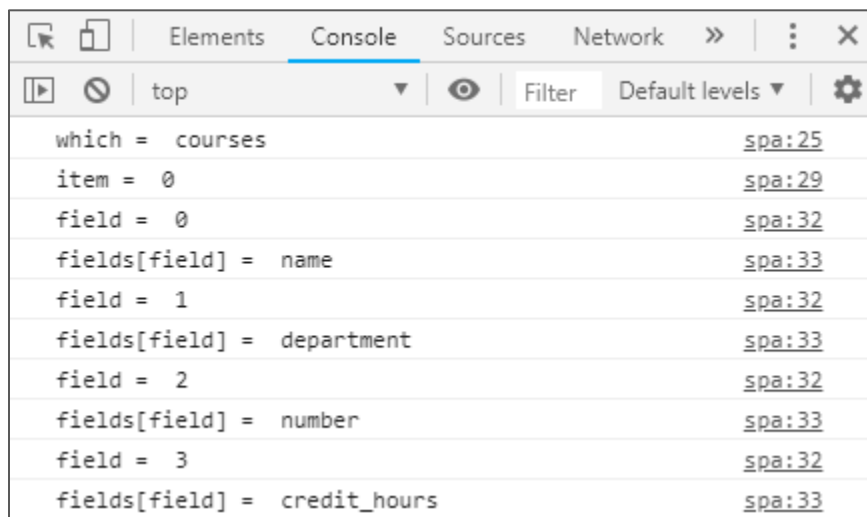
Now that we have a good idea of the basic structure of our data table in terms of HTML tag hierarchy, let's see if we can parse our JSON objects in such a way as to recreate our data tables in our SPA. First let's create our table headers.

- Let's add some more print statements to debug our way into finding the JSON field we want to access. Add the following lines in your script:

```
for (item in json) {  
  console.log("item = ", item);
```

```
for (field in fields) {  
  console.log("field = ", field);  
  console.log("fields[field] = ", fields[field]);
```

- If we navigate to the 'Courses' subpage of our SPA, we can see the table headers we need equal to the 'fields[field]' JSON object in the web console.



which = courses	spa:25
item = 0	spa:29
field = 0	spa:32
fields[field] = name	spa:33
field = 1	spa:32
fields[field] = department	spa:33
field = 2	spa:32
fields[field] = number	spa:33
field = 3	spa:32
fields[field] = credit_hours	spa:33

- First let's make sure to open and close our table before and after the outer 'for (item in json)' loop using the `<table>` tag:

```
tmp = which;  
$("#list").append("<h1>" + tmp.charAt(0).toUpperCase() + tmp.slice(1) + "</h1>");  
$("#list").append("<table>");
```

```
} // end for (item in json)  
  
$("#list").append("</table>");
```

- Let's see if we can *append* to our list object `fields[field]`, wrapped in a `<th>` table header tag. We'll want this outside of our main nested loop. Add the following block *before* the `for (item in json)` loop in your script:

```
head = "<thead>";
for (field in fields) {
  console.log("field = ", field);
  console.log("fields[field] = ", fields[field]);
  head += "<th>" + fields[field] + "</th>";
}
head += "</thead>";
$("#list").append(head);
```

- It's a bit cramped and not formatted very well, but we were able to create our table headers. We'll adjust the Cascading Style Sheet (CSS) formatting later on to adjust the margins and spacing of our SPA.

Students Courses Sections Enrollments

Courses

name	department	number	credit_hours
------	------------	--------	--------------

- Let's use the same technique we used previously to capitalize our table headers. Update the following line...

```
head += "<th>" + fields[field] + "</th>";
```

- to the following:

```
th = fields[field].charAt(0).toUpperCase() + fields[field].slice(1);
head += "<th>" + th + "</th>";
```

- Which gives us the following:

Courses

Name	Department	Number	Credit_hours
------	------------	--------	--------------

- We can clean up the underscore “_”, and the “id” from some of our headers as well:

```
th = (fields[field].charAt(0).toUpperCase() + fields[field].slice(1)).replace("_", " ");
```

```
th = (fields[field].charAt(0).toUpperCase() + fields[field].slice(1)).replace("_", " ").replace(" id", "");
```

- We now have a slightly improved table header:

Students	Courses	Sections	Enrollments
Sections			
Course	Semester	Number	Room number

Table Data

Now that we have a basic table header, let’s populate our table with data. Once again, we will iterate through our JSON objects and create an HTML tree structure containing the necessary `<tr>` table row and `<td>` table data tags.

- Here we use some useful print statements to find the right data items we need:

```
console.log("field = ", field);
console.log("fields[field] = ", fields[field]);
console.log("json[item][fields[field]] = ", json[item][fields[field]]);
```

```
fields[field] = name
json[item][fields[field]] = Astrorobotics
```

- Removing debugging statements leaves us with rather concise nested loop for parsing our JSON objects:

```
// table body
body = "<tbody>";
for (item in json) {
  row = "<tr>";
  data = "";
  for (field in fields) {
    data += "<td>" + json[item][fields[field]] + "</td>";
  }
  row += data + "</tr>";
  body += row;
}
body += "</tbody>";
$("#list").append(body);
// end table body
```

- The preceding loop producing the following table, which is quite close to what we want our SPA to look like:

Students Courses Sections Enrollments			
Courses			
Name	Department	Number	Credit hours
Astrorobotics	EGR	5600	9
Superfluid Dynamics	PHY	4500	7
Remote Viewing	PSY	2000	3
Intro to EBEs	EXO	2400	3

- Sections and Enrollments still display referenced “id’s” and not their relevant data:

Students Courses Sections Enrollments			
Enrollments			
Section	Student		
18	5		
18	3		
14	6		
14	7		
19	5		

JavaScript Functions

In order to display relevant Course, Student and Section info in our table, we’ll need to create a function that will further parse the JSON for a particular database item ID. First let’s create a simple function and see if we can capture the desired behavior.

- Create a simple JavaScript function, after our primary ‘fetch’ function:

```
var getById = function(which, id) {
    return "test";
};
```

- Now let’s add a branch to call our function within our inner loop.

```
for (field in fields) {
    if (fields[field] === "course_id") {
        data += "<td>" + getById(which, json[item][fields[field]]) + "</td>";
    }
    else {
        data += "<td>" + json[item][fields[field]] + "</td>";
    }
}
```

- We can see that our function gets called whenever a ‘course_id’ JSON field is parsed:

Students Courses Sections Enrollments			
Sections			
Course	Semester	Number	Room number
test	Spring 20801		260
test	Fall 2079	1	300
test	Fall 2079	1	149
test	Spring 20801		295

- We could create branches to capture the other ‘id’ fields in our JSON objects...

```
if (fields[field] === "course_id") {
  data += "<td>" + getId(which, json[item][fields[field]]) + "</td>";
}
else if (fields[field] === "student_id") {
  data += "<td>" + getId(which, json[item][fields[field]]) + "</td>";
}
else if (fields[field] === "section_id") {
  data += "<td>" + getId(which, json[item][fields[field]]) + "</td>";
}
else {
  data += "<td>" + json[item][fields[field]] + "</td>";
}
```

Students	Courses	Sections	Enrollments
----------	---------	----------	-------------

Enrollments

Section	Student
test	test
test	test
test	test
test	test
test	test

- ...or we could check for the substring ‘_id’ with the ‘includes’ function, which is far more concise:

```
for (field in fields) {
  if (fields[field].includes("_id")) {
    data += "<td>" + getId(which, json[item][fields[field]]) + "</td>";
  }
  else {
    data += "<td>" + json[item][fields[field]] + "</td>";
  }
}
```

- We achieve a similar result with fewer lines written:

Students	Courses	Sections	Enrollments
----------	---------	----------	-------------

Enrollments

Section	Student
test	test
test	test
test	test
test	test
test	test

- Let’s further parameterize our function call by calling based on *field* rather than *path*:

```
if (fields[field].includes("_id")) {
  data += "<td>" + getId(fields[field].replace("_id", "s"), json[item][fields[field]]) + "</td>";
}
```

- Now let’s test to make sure our function receives the field from which it was called correctly:

```
var getId = function(which, id) {
  return "test" + which;
};
```

- We can see that our function does indeed know where it was called from.

Students	Courses	Sections	Enrollments
----------	---------	----------	-------------

Enrollments

Section	Student
testsection	teststudent
testsection	teststudent
testsection	teststudent
testsection	teststudent
testsection	teststudent

- Now let's have our function retrieve the *name* field from the desired JSON object and return that value for our table (jQuery, 2018).

```
var getById = function(which, id) {
  var result = null;
  var value = null;
  $.get({
    url: "http://localhost:3000/" + which + ".json",
    dataType: "json",
    async : false,
    success: function(data) { result = data; }
  });
  for (cell in result) {
    if (result[cell].id == id) {
      var value = result[cell].name;
    }
  }
  return value;
};
```

- We see that that single 'if' is enough to fetch simple references to Course and Student names, yet it does not populate Section in Enrollments properly. It's referencing a 'section.name' field that doesn't exist.

Students	Courses	Sections	Enrollments
----------	---------	----------	-------------

Sections

Course	Semester	Number	Room	number
Astrorobotics	Spring 20	801		260
Superfluid Dynamics	Fall 2079	1		300
Remote Viewing	Fall 2079	1		149
Intro to EBEs	Spring 20801			295

Students	Courses	Sections	Enrollments
----------	---------	----------	-------------

Enrollments

Section	Student
undefined	James Howlett
undefined	Daria Morgendorffer
undefined	Charles Xavier
undefined	Max Eisenhardt
undefined	James Howlett

- Here we carefully handle a recursive call to our 'getById' function, and assemble a Section name string:

```
if (result[cell].id == id) {
  var value = result[cell].name;
  if (which === "sections") {
    console.log(result[cell]);
    var tmp = result;
    var index = cell;
    var course = getById("courses", result[cell].course_id);
    result = tmp;
    cell = index;
    value = course +
      " - " + result[cell].semester +
      " - Section " + result[cell].number +
      " - Room " + result[cell].room_number;
  }
}
```

- We have our desired result. Each sub-page now displays all relevant information in tabular format. The table and all data contained within it are created by our JavaScript function at runtime.

Students

Courses

Sections

Enrollments

Enrollments

Section	Student
Remote Viewing - Fall 2079 - Section 1 - Room 149	James Howlett
Remote Viewing - Fall 2079 - Section 1 - Room 149	Daria Morgendorffer
Astrorobotics - Spring 2080 - Section 1 - Room 260	Charles Xavier
Astrorobotics - Spring 2080 - Section 1 - Room 260	Max Eisenhardt
Intro to EBEs - Spring 2080 - Section 1 - Room 295	James Howlett

Bonus: CSS

Before we complete our work today, let's fix the spacing of our various HTML elements by specifying Cascading Style Sheet (CSS) options for our SPA.

- Add the following to your 'spa.html' file, inside the HTML header:

```
<style>
  button, h1, th, td, div { padding-left: 12px}
</style>
```


- Here we have our final Single-Page Application:

[Students](#)
[Courses](#)
[Sections](#)
[Enrollments](#)

Enrollments

Section	Student
Remote Viewing - Fall 2079 - Section 1 - Room 149	James Howlett
Remote Viewing - Fall 2079 - Section 1 - Room 149	Daria Morgendorffer
Astrorobotics - Spring 2080 - Section 1 - Room 260	Charles Xavier
Astrorobotics - Spring 2080 - Section 1 - Room 260	Max Eisenhardt
Intro to EBEs - Spring 2080 - Section 1 - Room 295	James Howlett

- Let's compare it to our previous version using multiple web pages:

Enrollments

Search for:

Section	Student	
Remote Viewing - Section 1 - Fall 2079	James Howlett	Show Edit Destroy
Remote Viewing - Section 1 - Fall 2079	Daria Morgendorffer	Show Edit Destroy
Astrorobotics - Section 1 - Spring 2080	Charles Xavier	Show Edit Destroy
Astrorobotics - Section 1 - Spring 2080	Max Eisenhardt	Show Edit Destroy
Intro to EBEs - Section 1 - Spring 2080	James Howlett	Show Edit Destroy

[New Enrollment](#)

- Now would be a great time to commit and push to your GitHub repository:

```
C:\CS390P-Web-Application-Architecture>git add .
```

```
C:\CS390P-Web-Application-Architecture>git commit -am "Assignment 5 final commit"
```

```
C:\CS390P-Web-Application-Architecture>git push origin master
```

Conclusion

We've come a long way as beginning web developers. We've gotten to know relational databases, web security, authentication and validation. Creating a Single-Page Application (SPA) allows us to present our hard work in a new way, without the interruption of load times between pages. The JavaScript Object Notation (JSON) API built within Rails allows us to easily parse JSON object data into usable JavaScript with which we can build our SPA. We were able to create JavaScript functions using jQuery that allow us to *parameterize* our sub-pages as we create them at runtime. Unfortunately, we weren't able to completely recreate our web application as a SPA, but we did learn just why JavaScript and jQuery are so popular. We have no doubt that a simple JavaScript function could recreate our Search bar, and that we could easily update our SPA to accommodate database Create/Update/Destroy (CUD) operations.

You can find all papers in this series, as well as complete working Rails 5.1 projects at my GitHub repository, available here:

<https://github.com/DarkLORDCOSMOS/CS390P-Web-Application-Architecture>

Bibliography

Beaty, S. (2018). *jQuery SPA* [GitHub repository]. Retrieved from

<https://github.com/drsjb80/SPAs/blob/master/jqspa.html>.

The jQuery Foundation (2018). *jQuery.ajax()* [website]. Retrieved from <http://api.jquery.com/jquery.ajax/>.

Zakas, N. (2014). *The principles of object-oriented JavaScript*. No Starch Press, Inc. San Francisco, CA.

Instructor Feedback

What was too difficult, too easy?

This assignment was of an appropriate difficulty. Choosing a framework and setting up one's basic SPA using the gracious template provided was perhaps the most difficult part. Debugging parsing logic is also quite difficult, though that goes into the 'fun' category too.

What would have made the learning experience better?

Perhaps this assignment could have been improved with a bit more time to finish updating our original Assignment 2 to incorporate Search and CUD operations into our SPA.

What did you learn?

I learned everything I know today about jQuery, JavaScript, and SPAs in this assignment.

How did you learn it?

I am grateful to the instructor for the example SPA files to get started. I learned a great deal from simply printing to the web console and making note of the result, saving small changes to my SPA and reloading it to make note of the result.