Models

Overview

The focus of this week should be on teaching the concepts of a model layer through the use of SQL and a few of its related tools and concepts. Students will be working with the module design pattern, in conjunction with multiple persistence tools, such as html5sql.js and our own custom abstraction layer, webDB.

These technologies and tools will be used to create a persistence layer, in the browser, that can be accessed dynamically through the use of JavaScript.

Students will focus on using CRUD operations to communicate with the database, along with modular JS patterns and locally sourced JSON data that can be accessed through AJAX.

The primary jQuery AJAX methods used during this week are as follows: `.ajax()`, `.get()`, `.getJSON()`.

Class 06: AJAX and WRRC

During this class, you will focus on teaching students the underlying concepts of how the web works. Starting with an introduction to the WRRC (web request response cycle), you will demonstrate how HTTP works and how to accomplish simple GET requests through the use of jQuery. Students will need to understand the 3 major parts of every request (url, method, headers) and the 3 major parts of every response (status, headers, body).

Class 06: Learning Objectives

- Identify when apps need persistence, to improve the UX and isolate the model logic in the code base.
- Explain how the browser uses the request-response (WRRC) cycle to render an HTML file or AJAX call.
- Analyze the localStorage object

Demo #1

Complete AJAX call

$.ajax() is the most configurable one

```
$.ajax({
    url: 'data/hackerIpsum.json',
    type: 'GET',
    dataType: 'json',
    success: function(data) {
      Article.loadAll(data);
      Article.all.forEach(function(a) {
        $('#articles').append(a.toHtml())
      });
      console.log(data);
    },
    error: function() {
      console.log("it failed");
    }
 });
```

Shorthand AJAX call

```
$.ajax('data/hackerIpsum.json', {
    success: function(data) {
    Article.loadAll(data);

  Article.all.forEach(function(a) {
    $('#articles').append(a.toHtml())
  });

  console.log(data);
  },
  error: function() {
    console.log("it failed");
  }
});
```

Shorthand version

It should be noted that all $.get(), $.post(), .load() are all just wrappers for $.ajax() as it's called internally.

$.get()

-- defaults type to get

```
$.get('data/hackerIpsum.json', function(data) {

  Article.loadAll(data);

  Article.all.forEach(function(a) {
    $('#articles').append(a.toHtml())
  });

  console.log(data);

});
```

$.getJSON()

-- defaults type to get

-- defaults dataType JSON

```
$.getJSON('data/hackerIpsum.json', function(data) {

    Article.loadAll(data);

  Article.all.forEach(function(a) {
    $('#articles').append(a.toHtml())
  });

  console.log(data);

});
```

$.load()

`.load()` is similar to `$.get()` but adds functionality which allows you to define where in the document the returned data is to be inserted. Therefore really only usable when the call only will result in HTML. It is called slightly differently than the other, global, calls, as it is a method tied to a particular jQuery-wrapped DOM element.

Example #1

```
<div id="divTestArea1"></div>


<script src="https://code.jquery.com/jquery-3.1.0.min.js" integrity="sha256-
cCueBR6CsyA4/9szpPfrX3s49M9vUU5BgtiJj06wt/s=" crossorigin="anonymous"></script>




<!-- Example #1 -->
<script type="text/javascript">
    $(function() {
        $("#divTestArea1").load("content.html");
    });
</script>
```

Example #2

A pretty cool trick is that you can actually pass a selector along with the URL, to only get a part of the page. In the first example, we loaded the entire file, but in the following example, we will only use the div, which contains the first sentence:

```
<!-- Example #2 -->
<div id="divTestArea2"></div>
<script type="text/javascript">
$(function() {
    $("#divTestArea2").load("content.html #divContent");
});
</script>
```

Deferred Methods

Chaining Promises

```
$.get('data/rawData.json', function(data) {
    Article.loadAll(data);

    Article.all.forEach(function(a) {
      $('#articles').append(a.toHtml())
    });

    console.log(data);

  })
  .done(function(){
    console.log("im done");
  })
  .always(function(){
    console.log("always!")
  });
```

Last Demo

```
Article.fetchAll = function() {
  if (localStorage.rawData) {
    // When rawData is already in localStorage,
    // we can load it by calling the .loadAll function,
    // and then render the index page (using the proper method on the
articleView object).
    var storedDataString = JSON.parse(localStorage.getItem('rawData'));
    Article.loadAll(storedDataString)//TODO: What do we pass in here to the
.loadAll function?

    articleView.initIndexPage()//(); //TODO: Change this fake method call to the
correct one that will render the index page.
    } else {
   // TODO: When we don't already have the rawData, we need to:
    // 1. Retrieve the JSON file from the server with AJAX (which jQuery method
is best for this?),
    $.ajax({
        type: 'GET',
        url: 'data/ipsumArticles.json',
          success: function (data) {
            // 2. Store the resulting JSON data with the .loadAll method,
            Article.loadAll(data);
            // 3. Cache it in localStorage so we can skip the server call next
time,
            localStorage.setItem('rawData', JSON.stringify(data));
            // 4. And then render the index page (perhaps with an articleView
method?).
            articleView.initIndexPage();

            console.log(data);
          }
      });
  }
}
```

Bonus if time

The ETag or entity tag is part of HTTP, the protocol for the World Wide Web. It is one of several mechanisms that HTTP provides for web cache validation, which allows a client to make conditional requests. This allows caches to be more efficient, and saves bandwidth, as a web server does not need to send a full response if the content has not cha

```
// This function will retrieve the data from either a local or remote source,
// and process it, then hand off control to the View.
```

```
Article.fetchAll = function() {
  function fetchFromDisk(){
    // DONE: When we don't already have the rawData, we need to:
    // 1. Retrieve the JSON file from the server with AJAX (which jQuery method
is best for this?),
    console.log('using ajax');
    $.getJSON( './data/hackerIpsum.json', function( data ) {
      // 2. Store the resulting JSON data with the .loadAll method,
      Article.loadAll(data);

      // 3. Cache it in localStorage so we can skip the server call next time,
      localStorage.setItem('rawData', JSON.stringify(data));

      // DONE. 4. And then render the index page (perhaps with an articleView
method?).
      articleView.initIndexPage();
    });
  }

  function fetchFromLocalStorage(){
    // When rawData is already in localStorage,
    // we can load it by calling the .loadAll function,
    // and then render the index page (using the proper method on the
articleView object).
    //DONE: What do we pass in here to the .loadAll function?
    console.log('using local storage');
    var rd = localStorage.getItem('rawData');
    var rdjson = JSON.parse(rd);
    Article.loadAll(rdjson);

    //DONE: Change this fake method call to the correct one that will render the
index page.
    articleView.initIndexPage();
  }

  $.ajax({
    url: './data/hackerIpsum.json',
    method: 'HEAD',
    success: function(data, message, xhr) {
      console.log('xhr', xhr);
      var etag = xhr.getResponseHeader('ETag');
      console.log('etag', etag);
      if (localStorage.etag){
        var locEtag = localStorage.getItem('etag');
        if (locEtag === etag && localStorage.rawData) {
          console.log('etag matches and in local storage');
          fetchFromLocalStorage();
        } else {
          fetchFromDisk();
        }
      } else {
        fetchFromDisk();
    }
      localStorage.setItem('etag', etag);
    }
  });
}
```