Controller Module

Overview

The focus of this week is the controller layer of an MVC application.

Through the use of page.js, students will be introduced to:

- client-side routing
- deployments and development environments,
- managing application state,
- and general OOP based modularity & refactoring.

Students will finalize their MVC apps by:

- completing our Single-Page App pattern with:
  - routing and controllers
  - integrating with 3rd-party REST APIs to display external data
  - deploying to production environments with a pre-existing server

Class 11: Routing and Controllers

Using page.js as a demonstration tool, you will guide students through route based functionality to create a more performant and modular single page web application.

These routes will be simple, with a single callback that hands off control to the controller, which will call the appropriate method(s) to render the page to the DOM.

Topic 1 - Review

Pair assignment Retrospective

- What went well?
- What was challenging?

Topic 2 - Routing and Controllers

Single-Page Apps

Lecture

Demo

PageJS

Page is a small client-side routing library for use with building single page applications (SPAs).

- It has a simple API which is inspired by Express.
- It utilizes the HTML5 history API under the hood
    - which is what allows you to build smooth user interfaces while still having linkable URLs for different pages of the app.

Routing

You saw how I can put routes, but we can also get data from routes:

Your routes can contain parameters, which you can assess through the context parameter to your route handler.

```
page('/user/:id', function(context){
  var userId = context.params.id;
  console.log('Loading details for user', userId);
});
```

Demo 1

See `Playground`

Demo 2

After you create `index.html` from `Playground_2`

Notes for `Playground_2`

Your routes can contain parameters, which you can assess through the context parameter to your route handler

```
page('/user/:id', function(context){
  var userId = context.params.id;
  console.log('Loading details for user', userId);
});
```

You can use wildcards as parameters too, in which case you will need to use array indexing to access the parameters:

```
page('/files/*', function(context){
    var filePath = context.params[0];
    console.log('Loading file', filePath);
});
```

A key difference between using a wildcard vs a named parameter is that a wildcard can match the character "/", while a named parameter cannot. In our file path example, using a wildcard allows filePath to contain arbitrarily nested subdirectories.

Another useful thing you can do with a wildcard is to define a fallback route:

```
page('*', function(){
    console.error('Page not found :(');
});
```