Comp 324/424 - Client-side Web Design

Spring Semester 2019 - Week 9

Dr Nick Hayward

DEV Week Assessment

Course total = 25%

- continue development of a web application
  - *built from scratch*
  - *HTML5, CSS, plain JavaScript...*
  - *continue design and development of initial project outline and design*
  - *working app (as close as possible...)*
  - ***NO** content management systems (CMSs) such as Drupal, Joomla, WordPress...*
  - ***NO** PHP, Python, Ruby, C# & .Net, Java, Go, XML...*
  - ***NO** CSS frameworks, such as Bootstrap, Foundation, Materialize...*
  - ***NO** CSS preprocessors such as Sass…*
  - ***NO** template tools such as Handlebars.js &c.*
  - *must implement data from either*
    - self hosted (MongoDB, Redis...)
    - APIs
    - cloud services (Firebase...)
    - **NO** SQL...e.g. (you may **NOT** use MySQL, PostgreSQL &c.)
- outline research conducted
- describe data chosen for application
- show any prototypes, patterns, and designs

DEV Week Demo

DEV week assessment will include the following:

- brief presentation or demonstration of current project work
  - *~ 5 to 10 minutes per group*
  - *analysis of work conducted so far*
  - *e.g. during semester & DEV week*
  - *presentation and demonstration*
  - *outline current state of web app*
  - *explain what works & does not work*
  - *show implemented designs since project outline & mockup*
  - *show latest designs and updates*

Further information on course website

- coursework

HTML5, CSS, & JS - example - part 1

Structure

- combine HTML5, CSS, and JavaScript, to create an example application
- outline of our project's basic directory structure

```
.
|- assets
|   |- images //logos, site/app banners - useful images for site's design
|   |- scripts //js files
|   |- styles  //css files
|- docs
|   |- json //any .json files
|   |- txt //any .txt files
|   |- xml //any .xml files
|- media
|   |- audio //local audio files for embedding & streaming
|   |- images //site images, photos
|   |- video //local video files for embedding & streaming
|- index.html
```

- each of the above directories can, of course, contain many additional sub-directories
  - *|– `images` may contain sub-directories for albums, galleries...*
  - *|– `xml` may contain sub-directories for further categorisation..*
  - *and so on...*

HTML5, CSS, & JS - example - part 2

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>travel notes - v0.1</title>
    <meta name="description" content="information on travel destinations">
    <meta name="author" content="ancientlives">
    <!-- css styles... -->
    <link rel="stylesheet" type="text/css" href="assets/styles/style.css">
  </head>
  <body>
    ...
    <!-- js scripts... -->
    <script type="text/javascript"      src="assets/scripts/jquery.min.js"
    <script type="text/javascript" src="assets/scripts/travel.js"></script
  </body>
</html>
```

- JS files at foot of body
  - *hierarchical rendering of page by browser - top to bottom*
  - *JS will now be one of the last things to load*
  - *JS files often large, slow to load*
  - *helps page load faster...*

HTML5, CSS, & JS - example - part 3

index.html - body

```
<body>
  <!-- document header -->
  <header>
    <h3>travel notes</h3>
    <p>record notes from various cities and placed visited...</p>
  </header>
  <!-- document main -->
  <main>
    <!-- note input -->
    <section class="note-input">
    </section>
    <!-- note output -->
    <section class="note-output">
    </section>
  </main>
  <!-- document footer -->
  <footer>
    <p>app's copyright information, additional links...</p>
  </footer>
  <!-- js scripts... -->
  <script type="text/javascript" src="assets/scripts/jquery.min.js"></scri
  <script type="text/javascript" src="assets/scripts/travel.js"></script>
</body>
```

HTML5, CSS, & JS - example - part 4

style.css

```
body {
  width: 850px;
  margin: auto;
  background: #fff;
  font-size: 16px;
  font-family: "Times New Roman", Georgia, Serif;
}
h3 {
  font-size: 1.75em;
}
header {
  border-bottom: 1px solid #dedede;
}
header p {
  font-size: 1.25em;
  font-style: italic;
}
footer p {
  font-size: 0.8em;
}
```

HTML5, CSS, & JS - example - part 5.1

travel.js

```
//overall app logic and loader...
function travelNotes() {
    "use strict";

    $(".note-output").html("<p>first travel note for Marseille...</p>");

};

$(document).ready(travelNotes);
```

- a simple JS function to hold the basic logic for our app
- call this function any reasonable, logical name
- in initial function, we set the `strict` pragma
- add an example call to the jQuery function, `html()`
  - *sets some initial note content*
- function `travelNotes()` loaded using the jQuery function `ready()`
  - *many different ways to achieve this basic loading of app logic*

HTML5, CSS, & JS - example - part 5.2

travel.js - plain JS

```
function travelNotes() {
  "use strict";

  // get a reference to `.note_output` in the DOM
  // n.b. these can be combined as well...
  let noteOutput = document.querySelector('.note-output');
  noteOutput.innerHTML = '<p>first travel note for Marseille...</p>';

}

// load app
travelNotes();
```

- DEMO 1 - travel notes - series 1

HTML5, CSS, & JS - example - part 6

add a note

- app's structure includes three clear semantic divisions of content
  - *`<header>`, `<main>`, and `<footer>`*
- `<main>` content category - create and add our notes for our application
- allow a user to create a new note
  - *enter some brief text, and then set it as a note*
- output will simply resemble a heading or brief description for our note
- add HTML element `<input>` to allow a user to enter note text
  - *new attributes in HTML5 such as autocomplete, autofocus, required, width...*
  - *set accompanying*

```
<h5>add note</h5>
<input>
```

```
<input type="text" value="add a note...">
```

HTML5, CSS, & JS - example - part 7

tidy up styling

- additional styles to create correct, logical separation of visual elements and content
- add a border to the top of our footer
  - *perhaps matching the header in style*
- update the box model for the `<main>` element
- add some styling for `<h5>` heading

```
h5 {
  font-size: 1.25em;
  margin: 10px 0 10px 0;
}
main {
  overflow: auto;
  padding: 15px 0 15px 0;
}
footer {
  margin-top: 5px;
  border-top: 1px solid #dedede;
}
```

HTML5, CSS, & JS - example - part 8

input update

```
<input><button>add</button>
```

```
.note-input input {
  width: 40%;
}
.note-input button {
  padding: 2px;
  margin-left: 5px;
  border-radius: 0;
  border: 1px solid #dedede;
  cursor: pointer;
}
```

- also update css for `input` and `button`
- remove button's rounded borders to match style of `input`
- match border for button to basic design aesthetics
- set cursor appropriate for a link style...
- DEMO 2 - travel notes - series I

HTML5, CSS, & JS - example - part 9.1

interaction - add a note

- added and styled our input and button for adding a note
- use jQuery to handle click event on button
- update `travel.js` file for event handler

```
//handle user event for `add` button click
$(".note-input button").on("click", function(e) {
  console.log("add button clicked...");
});
```

HTML5, CSS, & JS - example - part 9.2

interaction - add a note - plain JS

```
let addNoteBtn = document.getElementById('add-note');
addNoteBtn.addEventListener('click', () => {
  console.log('add button clicked...');
});
```

HTML5, CSS, & JS - example - part 10.1

interaction - add a note - output

- update this jQuery code to better handle and output the text from the input field
- what is this handler actually doing?
  - *jQuery code has attached an event listener to an element in the DOM*
  - *referenced in the selector option at the start of the function*
  - *uses standard CSS selectors to find the required element*
- jQuery can select and target DOM elements using standard CSS selectors
  - *then manipulate them, as required, using JavaScript*

```
//handle user event for `add` button click
$(".note-input button").on("click", function(e) {
  $(".note-output").append("<p>sample note text...</p>");
});
```

- output some static text to `note-output`

HTML5, CSS, & JS - example - part 10.2

interaction - add a note - output - plain JS

```
function travelNotes() {
  "use strict";

  // get a reference to `.note_output` in the DOM
  let noteOutput = document.querySelector('.note-output');
  // add note button
  let addNoteBtn = document.getElementById('add-note');

  // add event listener to add note button
  addNoteBtn.addEventListener('click', () => {
    // create p node
    let p = document.createElement('p');
    // create text node
    let noteText = document.createTextNode('sample note text...');
    // append text to paragraph
    p.appendChild(noteText);
    // append new paragraph and text to existing note output
    noteOutput.appendChild(p);
  });

}
```

- DEMO 3 - travel notes - series I

HTML5, CSS, & JS - example - part 11.1

interaction - add a note - output

```
//overall app logic and loader...
function travelNotes() {
    "use strict";

  //handle user event for `add` button click
  $(".note-input button").on("click", function(e) {
    //object for wrapper html for note
    var $note = $("<p>");
    //get value from input field
    var note_text = $(".note-input input").val();
    //set content for note
    $note.html(note_text);
    //append note text to note-output
    $(".note-output").append($note);
  });

};

$(document).ready(travelNotes);
```

HTML5, CSS, & JS - example - part 11.2

interaction - add a note - output - plain JS

```
function travelNotes() {
  "use strict";

  // get a reference to `.note_output` in the DOM
  let noteOutput = document.querySelector('.note-output');
  // add note button
  let addNoteBtn = document.getElementById('add-note');
  // input field for add note
  let inputNote = document.getElementById('input-note');

  addNoteBtn.addEventListener('click', () => {
    // create p node
    let p = document.createElement('p');
    // get value from input field for note
    let inputVal = inputNote.value;
    // create text node
    let noteText = document.createTextNode(inputVal);
    // append text to paragraph
    p.appendChild(noteText);
    // append new paragraph and text to existing note output
    noteOutput.appendChild(p);
  });

}
```

- DEMO 4 - travel notes - series 1

HTML5, CSS, & JS - example - part 12.1

interaction - add a note - clear input

```
//overall app logic and loader...
function travelNotes() {
    "use strict";

  //handle user event for `add` button click
  $(".note-input button").on("click", function(e) {
    //object for wrapper html for note
    var $note = $("<p>");
    //define input field
    var $note_text = $(".note-input input");
    //conditional check for input field
    if ($note_text.val() !== "") {
    //set content for note
    $note.html($note_text.val());
    //append note text to note-output
    $(".note-output").append($note);
    $note_text.val("");
  }
  });

};

$(document).ready(travelNotes);
```

HTML5, CSS, & JS - example - part 12.2

interaction - add a note - clear input - plain JS

```
function travelNotes() {
  "use strict";

  // get a reference to `.note_output` in the DOM
  let noteOutput = document.querySelector('.note-output');
  // add note button
  let addNoteBtn = document.getElementById('add-note');
  // input field for add note
  let inputNote = document.getElementById('input-note');

  // add event listener to add note button
  addNoteBtn.addEventListener('click', () => {
    // create p node
    let p = document.createElement('p');
    // get value from input field for note
    let inputVal = inputNote.value;

    // check input value
    if (inputVal !== '') {
      // create text node
      let noteText = document.createTextNode(inputVal);
      // append text to paragraph
      p.appendChild(noteText);
      // append new paragraph and text to existing note output
      noteOutput.appendChild(p);
      // clear input text field
      inputNote.value = '';
    }
  });

}
```

- DEMO 5 - travel notes - series 1

HTML5, CSS, & JS - example - part 13.1

interaction - add a note - keyboard listener

- need to consider how to handle keyboard events
- listening and responding to a user hitting the return key in the input field
- similar pattern to user click on button

```
$(".note-input input").on("keypress", function (e) {
  if (e.keyCode === 13) {
    ...do something...
  }
});
```

- need to abstract handling both button click and keyboard press
- need to be selective with regard to keys pressed
- add a conditional check to our listener for a specific key
- use local variable from the event itself, eg: e, to get value of key pressed
- compare value of e against key value required

HTML5, CSS, & JS - example - part 13.2

interaction - add a note - keyboard listener - plain JS

```
// add event listener for keypress in note input field
inputNote.addEventListener('keypress', (e) => {
  // check key pressed by code - 13 - return
  if (e.keyCode === 13) {
    console.log('return key pressed...');
  }
});
```

- example recording keypresses
  - *Demo Editor*

HTML5, CSS, & JS - example - part 14

interaction - add a note - abstract code

- need to create a new function to abstract
  - *creation and output of a new note*
  - *manage the input field for our note app*
- moving logic from button click function to separate, abstracted function
- then call this function as needed
  - *for a button click or keyboard press*
  - *then create and render the new note*

```
//manage input field and new note output
function createNote() {
  //object for wrapper html for note
  var $note = $("<p>");
  //define input field
  var $note_text = $(".note-input input");
  //conditional check for input field
  if ($note_text.val() !== "") {
  //set content for note
  $note.html($note_text.val());
  //append note text to note-output
  $(".note-output").append($note);
  $note_text.val("");
  }
}
```

HTML5, CSS, & JS - example - part 15.1

interaction - add a note - travel.js

```
//overall app logic and loader...
function travelNotes() {
  "use strict";

  //manage input field and new note output
  function createNote() {
    //object for wrapper html for note
    var $note = $("<p>");
    //define input field
    var $note_text = $(".note-input input");
    //conditional check for input field
    if ($note_text.val() !== "") {
    //set content for note
    $note.html($note_text.val());
    //append note text to note-output
    $(".note-output").append($note);
    $note_text.val("");
    }
  }

  //handle user event for `add` button click
  $(".note-input button").on("click", function(e) {
    createNote();
  });

  //handle user event for keyboard press
  $(".note-input input").on("keypress", function(e){
    if (e.keyCode === 13) {
      createNote();
    }
  });
};
$(document).ready(travelNotes);
```

HTML5, CSS, & JS - example - part 15.2

interaction - add a note - plain JS

```
function travelNotes() {
  "use strict";

  // get a reference to `.note_output` in the DOM
  let noteOutput = document.querySelector('.note-output');
  // add note button
  let addNoteBtn = document.getElementById('add-note');
  // input field for add note
  let inputNote = document.getElementById('input-note');

  // add event listener to add note button
  addNoteBtn.addEventListener('click', () => {
      createNote(inputNote, noteOutput);
  });

  // add event listener for keypress in note input field
  inputNote.addEventListener('keypress', (e) => {
    // check key pressed by code - 13 - return
    if (e.keyCode === 13) {
      createNote(inputNote, noteOutput);
    }
  });

}

// create a note
// - input = value from input field
// - output = DOM node for output of new note
function createNote(input, output) {
    // create p node
    let p = document.createElement('p');
    // get value from input field for note
    let inputVal = input.value;
    // check input value
    if (inputVal !== '') {
      // create text node
      let noteText = document.createTextNode(inputVal);
      // append text to paragraph
      p.appendChild(noteText);
```

```
        // append new paragraph and text to existing note output
        output.appendChild(p);
        // clear input text field
        input.value = '';
    }
}

// load app
travelNotes();
```

- DEMO 6 - travel notes - series 1

HTML5, CSS, & JS - example - part 16

interaction - add a note - animate

- jQuery well-known for is its simple ability to animate elements
- many built-in effects available in jQuery
  - *build our own as well*
- to `fadeIn` an element, effectively it needs to be hidden first
- we hide our newly created note
- then we can set it to `fadeIn` when ready
- many additional parameters for jQuery's `fadeIn` function
  - *customise a callback*
  - *change the speed of the animation*
  - *and so on...*
- jQuery API - fadeIn

HTML5, CSS, & JS - example - part 17

interaction - add a note - animate js

```
//manage input field and new note output
function createNote() {
  //object for wrapper html for note
  var $note = $("<p>");
  //define input field
  var $note_text = $(".note-input input");
  //conditional check for input field
  if ($note_text.val() !== "") {
  //set content for note
  $note.html($note_text.val());
  //hide new note to setup fadeIn...
  $note.hide();
  //append note text to note-output
  $(".note-output").append($note);
  //fadeIn hidden new note
  $note.fadeIn("slow");
  $note_text.val("");
  }
}
```

- DEMO 7 - travel notes - series 1

HTML5, CSS, & JS - example - part 18

style and render notes

- we have some new notes in our app
- add some styling to help improve the look and feel of a note
- can set background colours, borders font styles...
- set differentiating colours for each alternate note
- allows us to try some pseudoclasses in the CSS
  - *specified paragraphs in the `note-output` section*

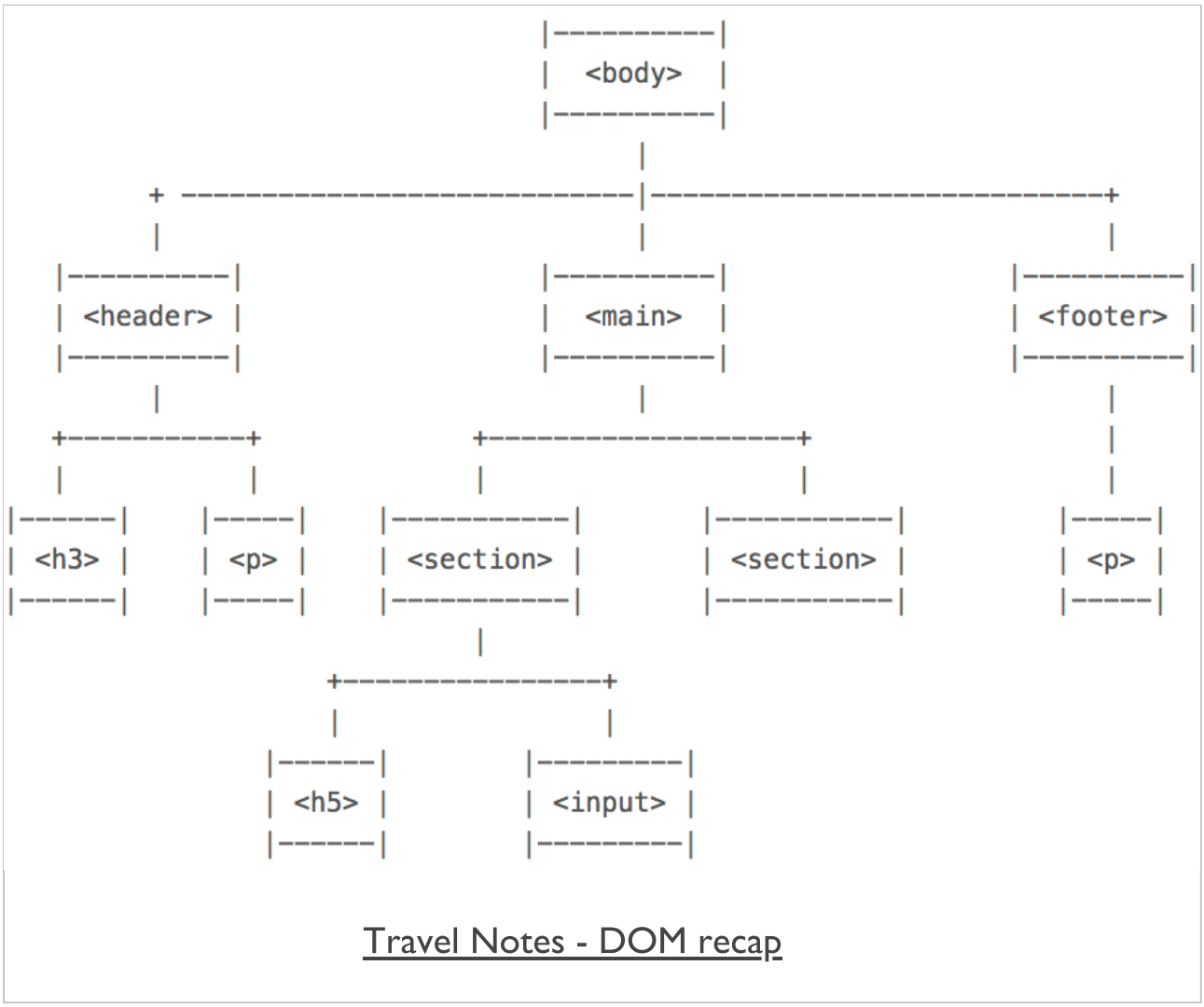
```
.note-output p:nth-child(even) {
  background-color: #ccc;
}
.note-output p:nth-child(odd) {
  background-color: #eee;
}
```

- DEMO 8 - travel notes - series 1

HTML5, CSS, & JS - final thoughts

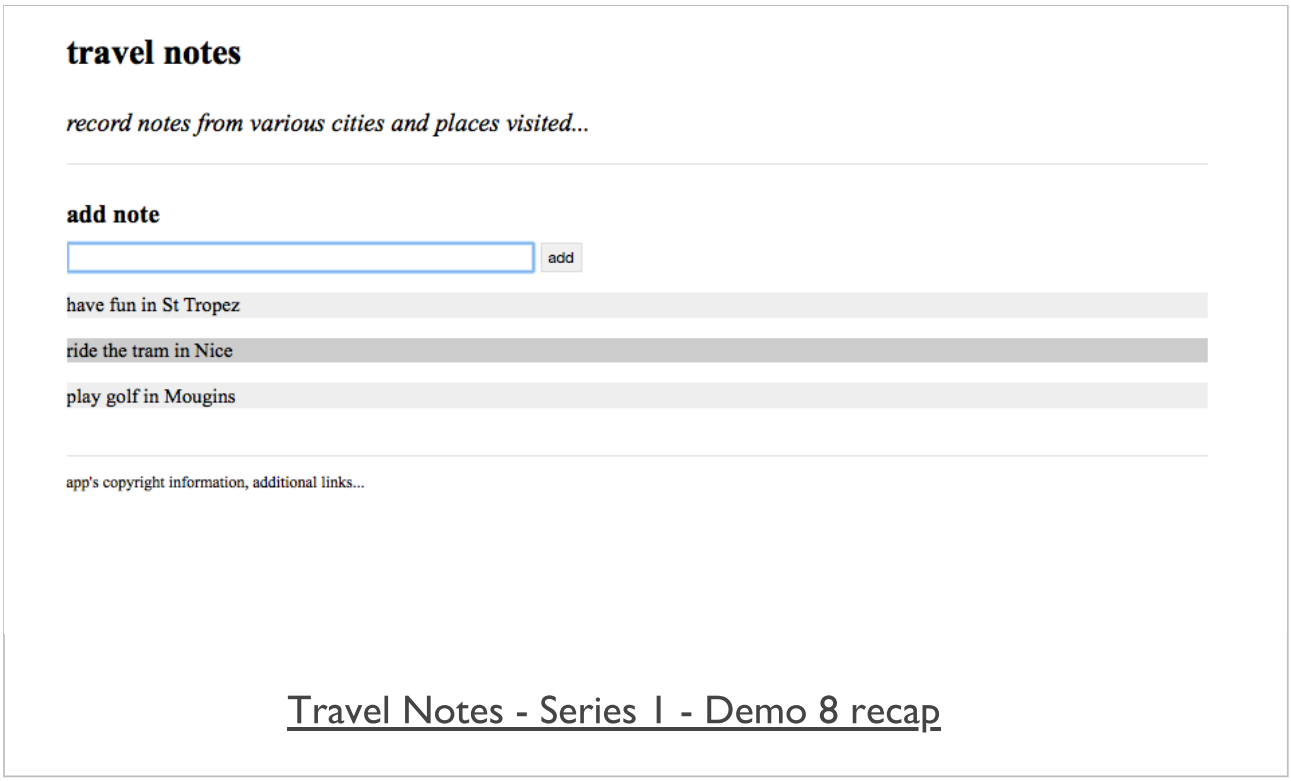
- a basic app that records simple notes
- many additional options we can add
- some basic functionality is needed to make it useful
  - *autosave - otherwise we lose our data each time we refresh the browser*
  - *edit a note*
  - *delete a note*
  - *add author information*
- additional functionality might include
  - *save persistent data to DB, name/value pairs...*
  - *organise and view collections of notes*
  - *add images and other media*
  - *local and APIs*
  - *add contextual information*
  - *again, local and APIs*
  - *structure notes, media, into collection*
  - *define related information*
  - *search, sort...*
  - *export options and sharing...*
- security, testing, design patterns

Image - HTML5, CSS, & JS - DOM recap



Travel Notes - DOM recap

Image - Travel Notes - Series 1 - recap



Travel Notes - Series 1 - Demo 8 recap

HTML5, CSS, & JS - example - add-ons

new features and add-ons...

- delete all notes
- delete a single note
- new event handlers
- additional styling

HTML5, CSS, & JS - example - part 1.1

delete option - all notes

- standard `remove()` function in jQuery

```
$("p").remove();
```

- option to **delete all** notes from `.note-output`
- add a new toolbar for note controls and options

```
<section class="note-controls">
  <button id="notes-delete">Delete all</button>
</section>
```

- then add some simple styling for this new toolbar

```
/* note controls */
.note-controls {
  margin: 10px 0 10px 0;
  padding: 2px;
  border-bottom: 1px solid #dedede;
  display: none;
}
/* simplify default button styles for note controls */
.note-controls button {
  padding: 2px;
  margin: 2px;
  border-radius: 0;
  border: 1px solid #dedede;
  cursor: pointer;
}
```

HTML5, CSS, & JS - example - part 1.2

delete option - all notes - plain js

```
// delete all notes button
let deleteAll = document.getElementById('notes-delete');

// add event listener for delete all notes...
deleteAll.addEventListener('click', () => {
    // get notes from DOM
    let notes = noteOutput.querySelectorAll('p');
    // loop through notes and remove a single note per iteration...
    for (let note of notes) {
        note.remove();
    }
});
```

HTML5, CSS, & JS - example - part 2.1

delete option - all notes

- note controls toolbar is hidden, by default in the CSS
- need some way to check its visibility as we add our notes
  - *no notes, then the toolbar is not required*

```
//check element visibility - expects single element relative to display:no
function checkVisible(element) {
  if (element.is(":hidden")) {
    element.fadeIn();
  }
}
```

- simply checking a passed element to see whether it is hidden
  - *then `fadeIn()` as necessary*
- can update this method later on to check hidden and visible
- call this function as required

```
checkVisible($(".note-controls"));
```

HTML5, CSS, & JS - example - part 2.2

delete option - all notes - plain js

- use `display` property to check node

```
// check visibility of passed node
function checkVisible(node) {
    // check passed node's current visibility
    if (node.style.display != 'block') {
        // show in DOM to allow fadeIn...
        node.style.display = 'block';
        // call fadeIn for node in DOM
        fadeIn(node);
    }
}
```

- & usage with a defined node

```
// define node to check...
let controls = document.getElementById('controls');
// call function
checkVisible(controls);
```

HTML5, CSS, & JS - example - part 2.3

delete option - all notes - plain js

- use `visibility` property to check node

```
// check visibility of passed node
function checkVisible(node) {
    // check passed node's current visibility
    if (node.style.visibility = 'hidden') {
    // show in DOM to allow fadeIn...
    node.style.display = 'block';
    node.style.visibility = 'visible';
    // call fadeIn for node in DOM
    fadeIn(node);
    }
}
```

HTML5, CSS, & JS - example - part 3

delete option - all notes

- add a note, the `.note-controls` toolbar is shown
  - ***delete all*** *button now becomes available to our users*

```
//handle deletion of all notes
$("#notes-delete").on("click", function(e) {
  var $note = $(".note-output p");
  $(this).parent().hide();
  $note.remove();
});
```

- creating a new handler for the click events on the `#notes-delete` button
- hides its own container, the notes toolbar
- then removes all of the notes, `p`, from the `.note-output` section

HTML5, CSS, & JS - example - part 4.1

JS code so far

```
//check element visibility - expects single element relative to display:no
function checkVisible(element) {
  if (element.is(":hidden")) {
    element.fadeIn();
  }
}
...
//handle deletion of all notes
$("#notes-delete").on("click", function(e) {
  var $note = $(".note-output p");
  $(this).parent().hide();
  $note.remove();
});
```

HTML5, CSS, & JS - example - part 4.2

JS code so far - plain JS

- hide parent node for controls...

```
// delete all notes button
let deleteAll = document.getElementById('notes-delete');

// add event listener for delete all notes...
deleteAll.addEventListener('click', () => {
    // hide parent controls node...
    deleteAll.parentNode.style.display = 'none';
    // get notes from DOM
    let notes = noteOutput.querySelectorAll('p');
    // loop through notes and remove a single note per iteration...
    for (let note of notes) {
        // remove single node
        note.remove();
    }
});
```

- DEMO 1 - travel notes - series 2

HTML5, CSS, & JS - example - part 5

delete option - all notes

- still making an assumption notes exist in the note-output section
- add an additional function to check element exists in the DOM or not
- use `length` property - plain JS & jQuery

```
$("p").length
```

- new function for checking elements in the DOM - plain JS & jQuery

```
//check elements exists
function checkExist(element) {
  if (element.length) {
    return true;
  } else {
    return false;
  }
}
```

HTML5, CSS, & JS - example - part 6.1

delete option - all notes

- updated delete all notes option to include check for notes
- call `checkExist()` function in conditional statement

```
//handle deletion of all notes
$("#notes-delete").on("click", function(e) {
  //set note selector
  var $note = $(".note-output p");
  //check $note exists
  if (checkExist($note) === true) {
    //hide note-controls
    $(this).parent().hide();
    //remove all notes
    $note.remove();
  }
});
```

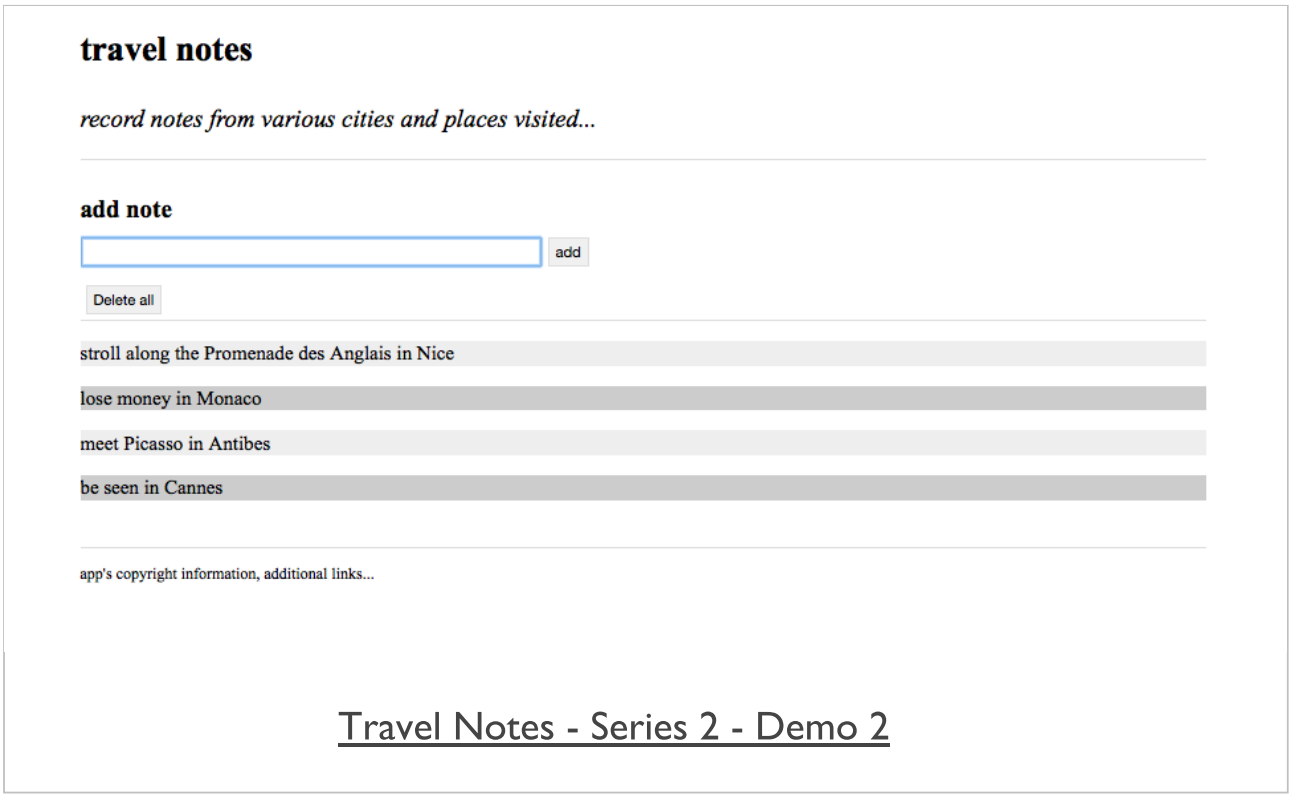
HTML5, CSS, & JS - example - part 6.2

delete option - all notes - plain JS

```
// add event listener for delete all notes...
deleteAll.addEventListener('click', () => {
    // get notes from DOM
    let notes = noteOutput.querySelectorAll('p');
    // check notes in DOM
    if (checkExist(notes) === true) {
        // hide parent controls node...
        deleteAll.parentNode.style.display = 'none';
        // loop through notes and remove a single note per iteration...
        for (let note of notes) {
            // remove single node
            note.remove();
        }
    }
});
```

- DEMO 2 - travel notes - series 2

Image - Travel Notes - Series 2 - demo 2



Travel Notes - Series 2 - Demo 2

HTML5, CSS, & JS - example - part 7

delete option - per note

- consider adding a single delete option per note
- allowing a user to selectively delete their chosen note
  - *regardless of hierarchical position within the* `.note-output` *section*
- design decisions for such an option might include
  - *do we offer a selection option, such as checkboxes, to select one or more delete items*
  - *perhaps a single delete button per note*
  - *a drag and drop to delete option*
  - *there are many different ways to present and use this option*
- programmatically follow a similar pattern for deletion of the note
- three jQuery functions can help us remove elements from a document
  - `remove()`
  - `detach()`
  - `replaceWith()`

jQuery - removing elements - quick overview

- used `remove()` function with delete all notes
  - *best used to remove elements permanently from a document*
  - *will **unbind** any attached event handlers for elements being removed*
  - *will return reference to removed elements, but not the original bound events*
- `detach()` often used for any temporary removal requirements
  - *eg: update a lot of the DOM, detach affected elements, then insert later...*
  - *retains its event handlers, and we can add these elements later*

```
$("p").detach();
```

- then append the attached elements as required

```
var $detachP = $("p").detach();
$detachP.appendTo("#detached");
```

- `replaceWith()` replaces an element, or group of elements, with passed element
- event handlers for the replaced elements are unbound

```
var $replacedP = $(".note-output p").first().replaceWith("<p>replaced...</
```

HTML5, CSS, & JS - example - part 8.1

delete option - per note

- simply need to delete the selected note
  - *use the same `remove()` function for single and all notes*
- add option per note to allow user to delete a required note
- add a delete button for each note
  - *add programmatically with each new note*

```
function createButton(buttonClass, buttonText) {
  var $button = $('<button class="'+buttonClass+'">'+buttonText+'</button>
  return $button;
}
```

- new function allows us to create simple buttons as required
  - *a specified class and button text passed as parameters*
  - *use function to build required delete button in `createNote()` function*

```
//create delete button
var $delete_button = createButton("note-delete", "delete");
```

HTML5, CSS, & JS - example - part 8.2

delete option - per note - plain js

```
// create button element - pass class and text
function createButton(btnClass, btnTxt) {
    // create button node
    let btnNode = document.createElement('button');
    // create button text node
    let btnTxtNode = document.createTextNode(btnTxt);
    // set attribute on button node
    btnNode.setAttribute('class', btnClass);
    // append text to button
    btnNode.appendChild(btnTxtNode);
    // return new button node with text and attribute...
    return btnNode;
}
```

- then call as required,

```
// create delete button for note
let delButton = createButton('note-delete', 'delete');
```

HTML5, CSS, & JS - example - part 9.1

delete option - per note

- append delete option to note
  - *before adding note to the DOM in `createNote` function*

```
function createNote() {
    ...
    //set content for note
    $note.html($note_text.val());
    //append delete button to each note
    $note.append($delete_button);
    ...
}
```

HTML5, CSS, & JS - example - part 9.2

delete option - per note - plain js

```
function createNote(input, output) {
  // get value from input field for note
    let inputVal = input.value;

    // check input value
    if (inputVal !== '') {
    // create p node
    let p = document.createElement('p');
        // create delete button for note
        let delButton = createButton('note-delete', 'delete');
        // prepend button to note
        p.prepend(delButton);
        // create text node
        let noteText = document.createTextNode(inputVal);
        // append text to paragraph
        p.appendChild(noteText);
        // append new paragraph and text to existing note output
        output.appendChild(p);
        // call custom animation for fade in...
        //fadeIn(p);
        // clear input text field
        input.value = '';
    }

    let controls = document.getElementById('app-controls');
    checkVisible(controls);
}
```

HTML5, CSS, & JS - example - part 10

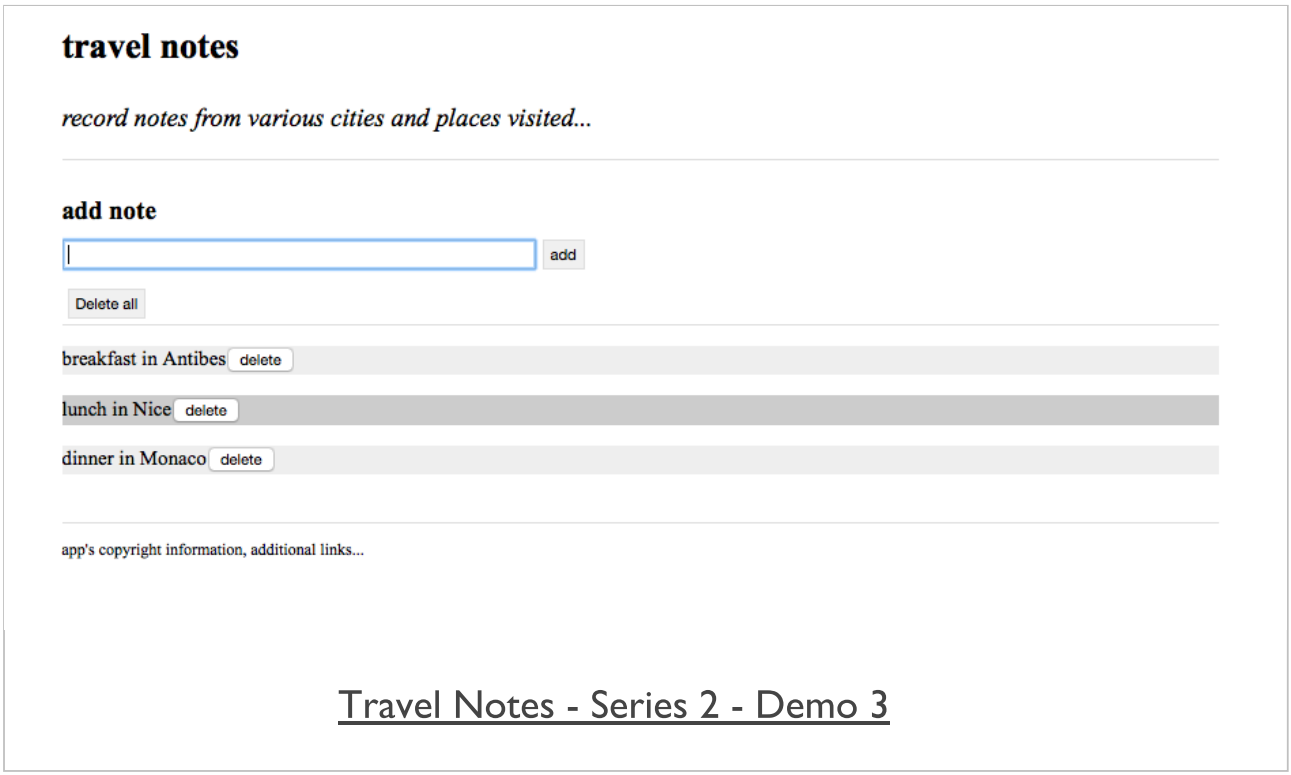
delete option - per note

- with jQuery
  - *need to bind a click event to the dynamically created delete note button*
    - plain JS option simpler
- delete button is being added to the DOM dynamically
  - *need to add handler for single note deletion event to existing DOM element*
  - *add handler to parent* `.note-output` *and then new* `button.note-delete`

```
$(".note-output").on("click", "button.note-delete" , function() {
  //delete parent note
  $(this).parent().remove();
  //set note selector
  var $note = $(".note-output p");
  //check for empty notes, and then remove note-controls
  if (checkExist($note) === false) {
    //hide note-controls
    $(".note-controls").hide();
  }
});
```

- DEMO 3 - travel notes - series 2 - jQuery

Image - Travel Notes - Series 2 - demo 3



Travel Notes - Series 2 - Demo 3

HTML5, CSS, & JS - example - part 11

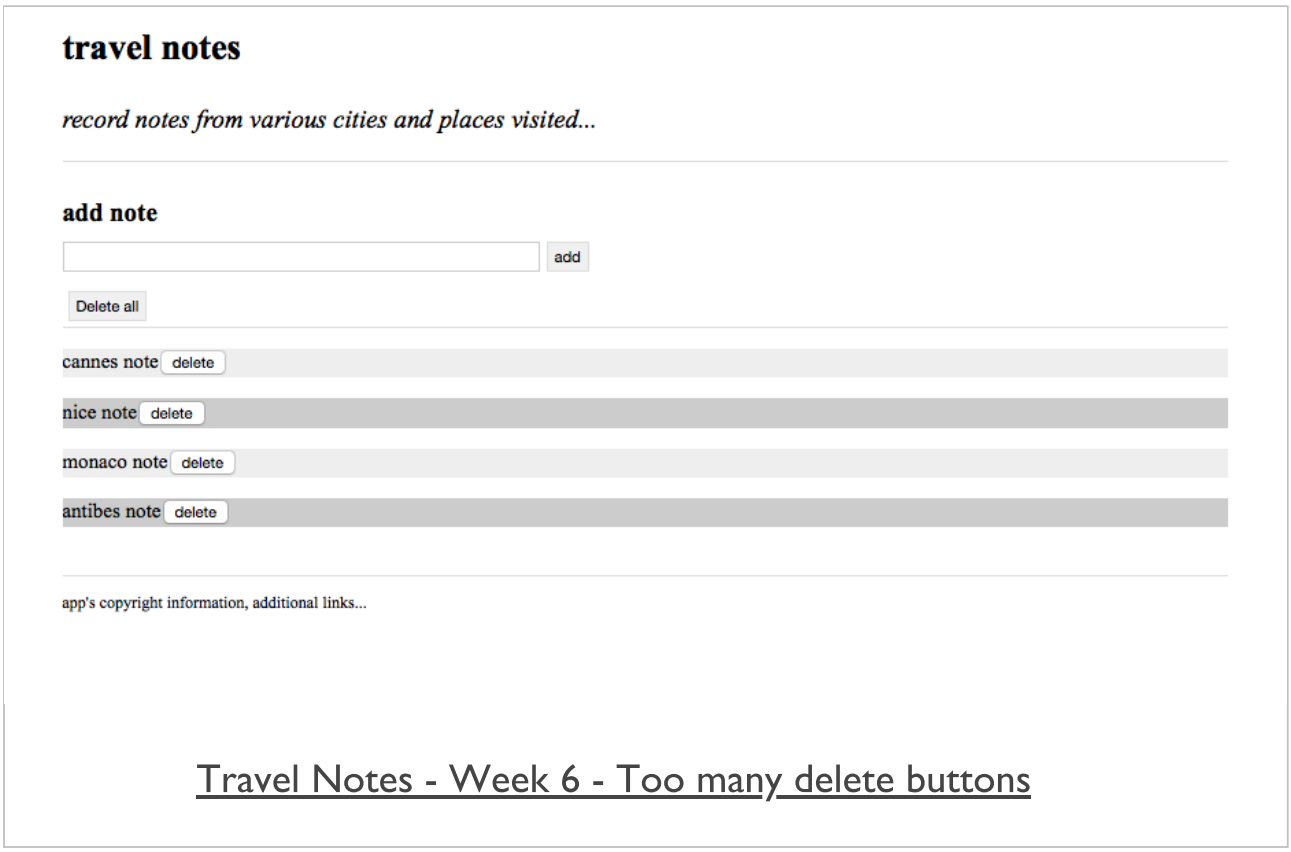
delete option - per note

- now allow our users to delete a single note
- single note option is awkward at the moment
- simply allow a user to either mouseover or select a note to show additional options
  - *showing the available delete button*
- enable a user to select their note of choice
  - *need to bind a click event to a note*

```
//handle click event per note
$(".note-output").on("click", "p", function() {
...
});
```

- user selects a note
  - *no check for previous other visible delete buttons*
  - *ensure only delete button for selected note is shown*

Image - HTML5, CSS, & JS - too many delete buttons



Travel Notes - Week 6 - Too many delete buttons

HTML5, CSS, & JS - example - part 12.1

delete option - per note

- return to our earlier function, `checkVisible()`
- modify to allow better abstraction and usage
- modify to test for visibility
  - *then simply return a boolean value*

```
//check element visibility - expects single element relative to display:no
function checkVisible(element) {
  //check if element is hidden or not
  if (element.is(":hidden")) {
    return true;
  } else {
    return false;
  }
}
```

- also need to modify check for the `.note-controls` in `createNote()` function

```
...
//check visibility of note controls
if (checkVisible($(".note-controls")) === true) {
  $(".note-controls").fadeIn();
}
...
```

HTML5, CSS, & JS - example - part 12.2

delete option - per note - plain js

- note delete button

```
// add delete button for current note
// use anonymous FN instead of arrow FN
// this binds to clicked DOM node
delButton.addEventListener('click', function () {
    console.log('note delete...', this.parentNode);
    this.parentNode.remove();
});
```

- note delete button with check for notes
- no notes - hide *delete all* option

```
// add delete button for current note
// use anonymous FN instead of arrow FN
// this binds to clicked DOM node
delButton.addEventListener('click', function () {
    console.log('note delete...', this.parentNode);
    this.parentNode.remove();
    // get notes from DOM
    let notes = output.querySelectorAll('p');
    if (checkExist(notes) === false) {
        controls.style.display = 'none';
    }
});
```

- DEMO 3 - travel notes - series 2 - plain JS

HTML5, CSS, & JS - example - part 13.1

delete option - per note

- updated handler for note selection now checks for visible delete buttons

```
//handle click event per note
$(".note-output").on("click", "p", function() {
  //check if other delete buttons visible
  if (checkVisible($("button.note-delete")) === true) {
    $("button.note-delete").hide();
  }
  $(this).children("button.note-delete").show();
});
```

- bind handler for the user clicking on a note
- check whether other delete buttons are visible on any other notes
  - *if visible, we can simply hide these delete buttons*
  - *then show the delete option for the currently selected note*
- later abstract this function to handle other options associated with each note

HTML5, CSS, & JS - example - part 13.2

delete option - per note - plain JS

- check for current delete buttons per note
  - *hide each delete button*
  - *then, show delete button for current note...*

```
// click listener for note
p.addEventListener('click', function() {
    // get notes delete buttons from DOM
    let delBtns = output.querySelectorAll('.note-delete');
    if (checkExist(delBtns) === true) {
        for (let btn of delBtns) {
            btn.style.display = 'none';
        }
    }
    this.querySelector('.note-delete').style.display = 'inline';
});
```

- DEMO 4 - travel notes - series 2
  - *jQuery*
  - *plain JS*

HTML5, CSS, & JS - example - part 14

style note(s)

- add some additional styling to our notes
  - *start with some changes to the design of each note*
  - *then considered the overall* `.note-output` *section*
- remove styling for alternating notes, set uniform style per note

```
/* note paragraph output */
.note-output p {
  margin: 10px;
  padding: 10px;
  border: 1px solid #b1c4b1;
  cursor:pointer;
}
```

- need to add some styling for our delete button, and position it within each note

```
/* note delete button */
.note-output p button.note-delete {
  display: block;
  padding: 5px;
  margin: 5px 5px 10px 0;
  border-radius: 0;
  border: 1px solid #dedede;
  cursor: pointer;
}
```

HTML5, CSS, & JS - example - part 15

style note(s)

- add some styling for the button's hover pseudo-class
  - *acts as useful feedback to the user that the button is an active element*

```
.note-output p button.note-delete:hover {
  background-color: #aaa;
  color: #fff;
}
```

- also consider adding some similar feedback to our note
  - *a sign of active as the user moves their mouse cursor over each note*

```
/* note paragraph output hover */
.note-output p:hover {
  border: 1px solid #1a3852;
}
```

- DEMO 5 - travel notes - series 2
  - *jQuery*
  - *plain JS*

HTML5, CSS, & JS - example - part 16

style note(s)

- a couple of issues that still need to be fixed in the application
  - *first issue is lack of consistency in styling our buttons*
- fixed by abstracting our CSS styling for a default button
  - *specific button styles can be added later*

```
/* default button style */
button {
  padding: 2px;
  margin: 2px;
  border-radius: 0;
  border: 1px solid #dedede;
  cursor: pointer;
}
```

- removed the need for a ruleset to style the button for
  - *adding a note, delete all notes, and the single delete button per note*

HTML5, CSS, & JS - example - part 17

style note(s)

- also create a default ruleset for a button hover pseudo-class
  - *again reducing our need for repetition in the stylesheet*

```
/* default button hover style */
button:hover {
  background-color: #aaa;
  color: #fff;
}
```

- iterative development is fine
  - *continue to abstract styles, overall design, and logic as we develop an application*

HTML5, CSS, & JS - example - part 18

style note(s)

- second issue is the expected interaction with each note
  - *issue is simply that a user cannot choose to remove this option*
- should be able to toggle its view and options
- update interaction by modifying handler for click event on a note
  - ***NB:*** `toggle()` *for events was removed in jQuery 1.9*
  - *build our own*

```
//handle click event per note
$(".note-output").on("click", "p", function() {
  //check if other delete buttons visible
  if (checkVisible($("button.note-delete")) === true) {
    //set all siblings to active=false to ensure checks are correct
    $(this).siblings().attr("active", "false");
    $("button.note-delete").hide();
  }
  //then handle click event for current note
  if (!$(this).attr("active") || $(this).attr("active") === "false") {
    $(this).attr("active", "true");
    $(this).children("button.note-delete").show();
  } else if ($(this).attr("active") === "true") {
    $(this).attr("active", "false");
    $(this).children("button.note-delete").hide();
  }
});
```

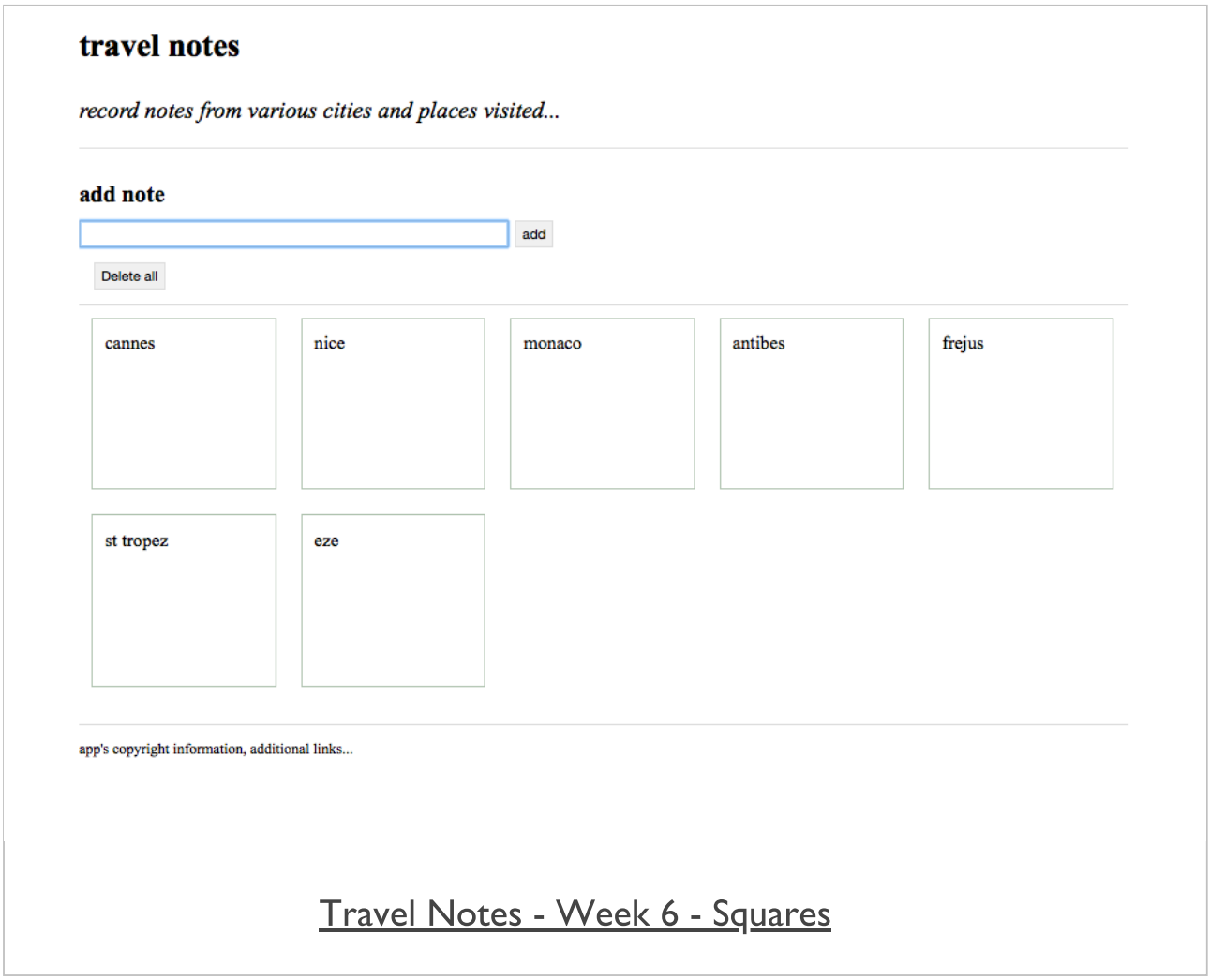
- DEMO 6 - travel notes - series 2

HTML5, CSS, & JS - example - part 19

a few extras to consider...

- alternative layouts
  - *grid*
  - *squares*
  - *snippet view*
  - *table*
  - *lists...*
- notifications
- snippets with expansion
- split views
  - *note snippet with contextual/media per note...*
- drag and drop delete
- filters
- sort options
- tags
- much, much more...

Image - Square notes - a bit of fun



Travel Notes - Week 6 - Squares

- DEMO - travel notes - squares

JS Objects - quick recap - part 1

- important JavaScript primitive
  - *one of the most frequently used as well*
- created with curly braces,

```
var object1 = {
  "a":"nine",
  "b":"ten"
};
```

- access internal variables of this object using the dot `.` operator

```
console.log(object1.a);
```

- update the value of an internal variable

```
object1.a = "amelia";
```

JS Objects - quick recap - part 2

- also create an empty variable, and then assign values as necessary

```
var object1 = {};
```

- an object can contain variables with values of different types
- store variables in an object with types such as strings, arrays, and even other objects
- function variables behave just like any other variables in JavaScript
  - *we can also store them in our objects as needed*

```
var $a = $("p");
$a.hide();
```

- simply attach a function to a jQuery object

JSON - quick recap

- a JSON object is effectively a JavaScript object
  - *contained within curly braces*

```
{
  "country":"France",
  "city":"Marseille"
}
```

- objects can contain multiple name/value pairs
- object stored in the form of a string
- to send a JS object
  - *create it in the application's code*
  - *then convert it to a string*
  - *finally use it as required*
- a lot of the AJAX is abstracted to JavaScript libraries

JSON - pros and cons

useful pros

- more concise, less verbose than XML and HTML
  - *potentially faster execution of data...*
- regularly used with JavaScript
  - *includes good support*
- language agnostic, interoperability
  - *can be used with many different programming languages*
- can also be called from many different domains
  - *eg: JSON-P...*

some cons

- may present security risk
  - *malicious content due to JavaScript XSS*
  - *need to verify source for JSON...*
- syntax is precise, unforgiving

JS and JSON - functions

- creating some JSON string is easy enough
- also easily create a JSON string from a JavaScript object
  - *and vice-versa*
- use the JavaScript `stringify` function

```
var jsonObject1 = JSON.stringify(object1);
console.log(jsonObject1);
```

- similarly parse a JSON string to a JS object

```
var object2 = JSON.parse(jsonObject1);
console.log(object2);
```

AJAX and JSON - part 1

intro

- AJAX is a simple way to load data
  - *often new or updated data*
  - *into a current page without having to refresh the browser window*
- common form of data for work with AJAX is JSON
- many common usage scenarios and examples for AJAX
  - *autocomplete in forms*
  - *live filtering of search queries*
  - *real-time updates for content and data streams*
- also use AJAX to help us load data behind the scenes
  - *preparing content for our users before a specific request is received*
  - *helps to speed up page responses and data load times*
- AJAX uses an asynchronous model for processing requests
- user can continue to perform various tasks, queries, and work
  - *whilst the browser itself continues to load data*
- inherent benefit of AJAX should include
  - *a more responsive site, intuitive usage and interface experience*

AJAX and JSON - part 2

asynchronous model

- traditional synchronous model normally stops a page
  - *until it has loaded and processed a requested script*
- AJAX enables a browser to request data from the server
  - *without this synchronous pause in usage*
- AJAX's **asynchronous processing model**
  - *often known as **non-blocking***
  - *allows a page to load data and process user's interactions*
- server responds with the requested data
  - *an event will be fired by the browser*
  - *event can then call a function to process the data*
  - *often JSON, XML, or simply HTML*
- browser will use an **XMLHttpRequest** object to help handle these AJAX requests
- browser will not wait for a response

JSON and jQuery - get a file - part 1

initial setup

- try some AJAX with a JSON file

```
{
  "country":"France",
  "city":"Marseille"
}
```

- save this content to our `docs/json/trips.json` file
- run on a server, local or remote
  - *browser security restrictions for JavaScript*
  - *local server such as XAMPP, Python's SimpleHTTPServer, Node.js...*

```
python -m SimpleHTTPServer 8080
```

- initially use the `getJSON()` function to test reading this content

```
$.getJSON("docs/json/trips.json", function(trip) {
  console.log(trip);
});
```

- console output is expected JSON object

```
Object { country: "France", city: "Marseille" }
```

JSON and jQuery - get a file - part 2

test with site

- now use this return object to load our data as required within a site

```
//overall app logic and loader...
function loadJSON() {
  "use strict";

  $.getJSON("docs/json/trips.json", function(trip) {
    //element for trip data
    var $tripData = $("<p>");
    //add some content from json to element
    $tripData.html(trip.city + ", " + trip.country);
    //append content to .note-output section
    $(".note-output").append($tripData);
  });
};

$(document).ready(loadJSON);
```

- DEMO - AJAX 1 - AJAX - demo 1

JSON and jQuery - get a file - part 3

array for trips...

Whilst the previous example is useful, for our application we obviously need to store multiple trips. So, multiple countries, multiple cities, and so on. Therefore, we need to consider working with JSON arrays. We'll update our `trips.json` file as follows to test loading cities,

```
{
    "cities": [
        {
            "name": "Marseille",
            "region": "Provence-Alpes-Côte d'Azur"
        },
        {
            "name": "Paris",
            "region": "Île-de-France"
        }
    ]
}
```

JSON and jQuery - get a file - part 4

load an array for trips...

- update JavaScript to load array and set data as required

```
//overall app logic and loader...
function loadJSON() {
  "use strict";

  $.getJSON("docs/json/trips.json", function(trips) {
    //element for trip data
    var $cityData = $("<ul>");

    //iterate over cities array - trips.cities...
    var $cities = trips.cities;
    $cities.forEach(function (item) {
      var $city = $("<li>");
      $city.html(item.name + " in the region of " + item.region);
      $cityData.append($city);
    })
    //append list to .note-output
    $(".note-output").append($cityData);
  });
};

$(document).ready(loadJSON);
```

- DEMO - AJAX 2 - AJAX - demo 2

Ajax, JSON & jQuery - part 1

jQuery Deferred

- jQuery provides a useful solution to the escalation of code for asynchronous development
- known as the `$.Deferred` object
  - *effectively acts as a central despatch and scheduler for our events*
- with the **deferred** object created
  - *parts of the code indicate they need to know when an event completes*
  - *whilst other parts of the code signal an event's status*
- **deferred** coordinates different activities
  - *enables us to separate how we trigger and manage events*
  - *from having to deal with their consequences*

Ajax, JSON & jQuery - part 2

using deferred objects

- now update our AJAX request with **deferred** objects
- separate the asynchronous request
  - *into the initiation of the event, the AJAX request*
  - *from having to deal with its consequences, essentially processing the response*
- separation in logic
  - *no longer need a success function acting as a callback parameter to the request itself*
- now rely on `.getJSON()` call returning a **deferred** object
- function returns a restricted form of this **deferred** object
  - *known as a **promise***

```
deferredRequest = $.getJSON (
  "file.json",
  {format: "json"}
);
```

Ajax, JSON & jQuery - part 3

using deferred objects

- indicate our interest in knowing when the AJAX request is complete and ready for use

```
deferredRequest.done(function(response) {
  //do something useful...
});
```

- key part of this logic is the `done()` function
- specifying a new function to execute
  - *each and every time the event is successful and returns complete*
  - *our AJAX request in this example*
- **deferred** object is able to handle the abstraction within the logic
- if the event is already complete by the time we register the callback via the `done()` function
  - *our **deferred** object will execute that callback immediately*
- if the event is not complete
  - *it will simply wait until the request is complete*

Ajax, JSON & jQuery - part 4

handling errors with deferred objects

- also signify interest in knowing if the AJAX request fails
- instead of simply calling `done()`, we can use the `fail()` function
- still works with JSONP
  - *the request itself could fail and be the reason for the error or failure*

```
deferredRequest.fail(function() {
  //report and handle the error...
});
```

Ajax, JSON & jQuery - part 5

example

- add the option to read and write from a JSON file
- we'll use AJAX for these requests
- initially we can consider our application as follows
  - *read data from JSON file*
  - *load initial data to application*
- no edit features for now
- add edit features with DB

Ajax, JSON & jQuery - part 6

example - JSON

- test reading and loading JSON file and data
- ignore standard AJAX pattern
  - *passing two callbacks, success and error*
- use `deferred` and `promise`
- initial JSON for Travel Notes app

```
{
  "travelNotes": [{
    "created": "2015-10-12T00:00:00Z",
    "note": "a note from Cannes..."
  }, {
    "created": "2015-10-13T00:00:00Z",
    "note": "a holiday note from Nice..."
  }, {
    "created": "2015-10-14T00:00:00Z",
    "note": "an autumn note from Antibes..."
  }]
}
```

Ajax, JSON & jQuery - part 7

example - deferred

- start by submitting a query for the required JSON file
- then retain the deferred object we're using for tracking
- then indicate interest in knowing when AJAX request is complete

```
//load main app logic
function loadApp() {
  "use strict";

    var $deferredNotesRequest = $.getJSON (
      "docs/json/notes.json",
      {format: "json"}
    );

    $deferredNotesRequest.done(function(response) {
        console.log("tracking json...");
    });

};
$(document).ready(loadApp);
```

Ajax, JSON & jQuery - part 8

example - deferred

- `done()` method is the key part
- helps us specify the required logic to execute
  - *when the request is complete*
- if the given event has already completed as callback is registered via `done()`
  - *deferred object will execute required callback immediately*
- if not, it will simply wait until request is complete
- respond to an error
  - *add `fail()` method for errors handling and reporting*

Ajax, JSON & jQuery - part 9

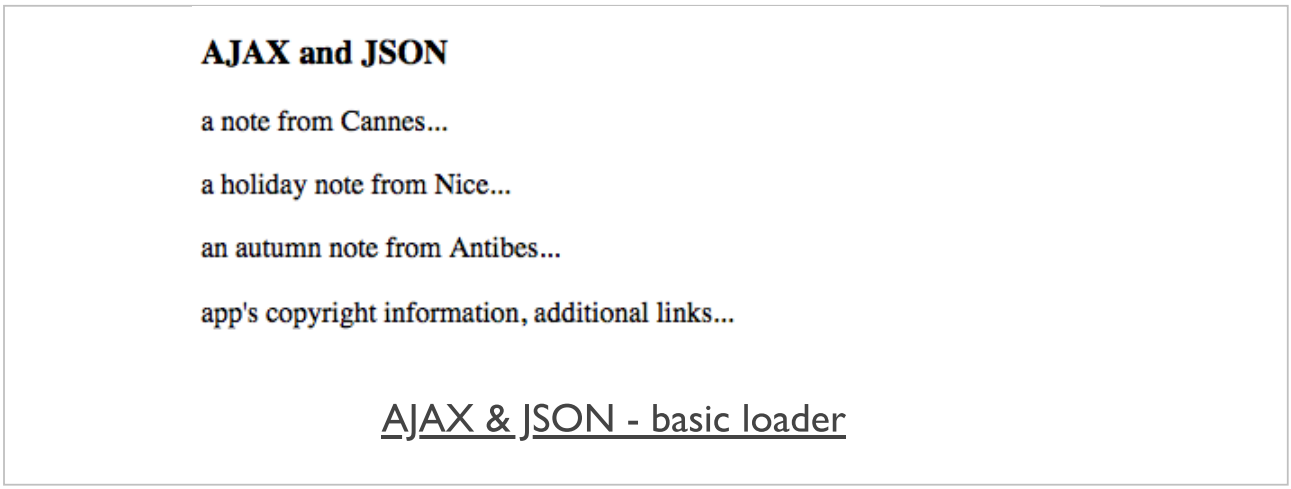
example - work with data

- returned data
  - *our response returns an object containing an array with notes*
- we could simply extract the required notes
  - *then append them to the DOM*

```
$deferredNotesRequest.done(function(response) {
    //get travelNotes
    var $travelNotes = response.travelNotes
    //process travelNotes array
    $travelNotes.forEach(function(item) {
      if (item !== null) {
        var note = item.note;
        //create each note's <p>
        var p = $("<p>");
        //add note text
        p.html(note);
        //append to DOM
        $(".note-output").append(p);
      }
    });
});
```

- DEMO - ajax & json basic loader

Image - HTML5, CSS, & JS - AJAX & JSON



Ajax, JSON & jQuery - part 10

example - work with data

- we can use simple deferred requests with our local JSON data
- with staggered API calls to data, need to use slightly modified approach
  - *digging through data layer by layer*
  - *submitting a request as one layer returns*
- we could now create a second deferred object
  - *use to track additional processing requests*
  - *stagger our requests to the API*
  - *ensuring we only request certain data as needed or available*
- also create multiple deferred objects to handle our requests and returned data
  - *allows us to respond accordingly within the application*

Ajax, JSON & jQuery - part 11

example - work with data

`resolve()`

- use this method with the deferred object to change its state, effectively to complete
- as we resolve a deferred object
  - *any **doneCallbacks** added with `then()` or `done()` methods will be called*
  - *these callbacks will then be executed in the order added to the object*
  - *arguments supplied to `resolve()` method will be passed to these callbacks*

`promise()`

- useful for limiting or restricting what can be done to the deferred object

```
function returnPromise() {
  return $.Deferred().promise();
}
```

- method returns an object with a similar interface to a standard `deferred` object
  - *only has methods to allow us to attach callbacks*
  - *does not have the methods required to resolve or reject deferred object*
- restricting the usage and manipulation of the deferred object
  - *eg: offer an API or other request the option to subscribe to the deferred object*
  - ***NB:** they won't be able to resolve or reject it as standard*

Ajax, JSON & jQuery - part 12

example - work with data

- still use the `done()` and `fail()` methods as normal
- use additional methods with these callbacks including the `then()` method
- use this method to return a new promise
  - *use to update the status and values of the deferred object*
  - *use this method to modify or update a deferred object as it is resolved, rejected, or still in use*
- can also combine promises with the `when()` method
  - *method allows us to accept many promises, then return a sort of master deferred*
- updated `deferred` object will now be resolved when all of the promises are resolved
  - *it will likewise be rejected if any of these promises fail*
- use standard `done()` method to work with results from all of the promises
  - *eg: could use this pattern to combine results from multiple JSON files*
  - *multiple layers within an API*
  - *staggered calls to paged results in a API...*

Ajax, JSON & jQuery - part 13

example - work with data

- now start to update our test AJAX and JSON application
  - *begin by simply abstracting our code a little*

```
function buildNote(data) {
  //create each note's <p>
  var p = $("<p>");
  //add note text
  p.html(data);
  //append to DOM
  $(".note-output").append(p);
}

//get the notes JSON
function getNotes() {
  //.get returns an object derived from a Deferred object - do not need ex
  var $deferredNotesRequest = $.getJSON (
  "docs/json/notes.json",
  {format: "json"}
  );
  return $deferredNotesRequest;
}
```

- DEMO - ajax & json abstract loader

Ajax, JSON & jQuery - part 14

example - work with data

- requesting our JSON file using `.getJSON()`
  - *we get a returned **promise** for the data*
- with a **promise** we can only use the following
  - *deferred object's method required to attach any additional handlers*
  - *or determine its state*
- our **promise** can work with
  - *`then`, `done`, `fail`, `always`...*
- our **promise** can't work with
  - *`resolve`, `reject`, `notify`...*

Ajax, JSON & jQuery - part 15

example - work with data

- one of the benefits of using **promises** is the ability to load one JSON file
  - *then wait for the results*
  - *then issue a follow-on request to another file*
  - ...
- a simple example of chained `then()` methods

```
getNotes().then(function(response1) {
  console.log("response1="+response1.travelNotes[2].note);
  $(".note-output").append(response1.travelNotes[2].note);
  return getPlaces();
}).then(function(response2) {
  console.log("response2="+response2.travelPlaces[2].place);
  $(".note-output").append(response2.travelPlaces[2].place);
});
```

- outputting a limited test result to the DOM and the console
- as we chain our `then()` methods
  - *pass returned results to next chained `then()` method...*
- DEMO - ajax & json deferred .then()

Demos

- Travel notes app - series 1
  - *DEMO 1 - travel notes - demo 1*
  - *DEMO 2 - travel notes - demo 2*
  - *DEMO 3 - travel notes - demo 3*
  - *DEMO 4 - travel notes - demo 4*
  - *DEMO 5 - travel notes - demo 5*
  - *DEMO 6 - travel notes - demo 6*
  - *DEMO 7 - travel notes - demo 7*
  - *DEMO 8 - travel notes - demo 8*
- Travel notes app - series 2 - jQuery
  - *DEMO 1 - travel notes - demo 1*
  - *DEMO 2 - travel notes - demo 2*
  - *DEMO 3 - travel notes - demo 3*
  - *DEMO 4 - travel notes - demo 4*
  - *DEMO 5 - travel notes - demo 5*
  - *DEMO 6 - travel notes - demo 6*
- Travel notes app - series 2 - plain JS
  - *DEMO 3 - travel notes - series 2 - plain JS*
  - *DEMO 4 - travel notes - series 2 - plain JS*
  - *DEMO 5 - travel notes - series 2 - plain JS*
- AJAX
  - *DEMO 1 - AJAX - demo 1*
  - *DEMO 2 - AJAX - demo 2*
- AJAX and JSON - jQuery
  - *abstract code for load a JSON file*
  - *load a JSON file*
  - *test deferred .then()*

Resources

- jQuery
  - *jQuery*
  - *jQuery API*
  - *jQuery - deferred*
  - *jQuery - .getJSON()*
  - *jQuery - JSONP*
  - *jQuery - promise*
- MDN
  - *MDN - JS*
  - *MDN - JS Const*
  - *MDN - JS - Iterators and Generators*
  - *MDN - JS Objects*