Comp 424 - Client-side Web Design

Fall Semester 2016 - Week 10

Dr Nick Hayward

Contents

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 I

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...

- need to store multiple trips
  - *multiple countries, multiple cities, and so on...*
- need to work with JSON arrays
  - *update `trips.json` file for 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

HTML5, CSS, & JS - travel notes

a few extras to consider...

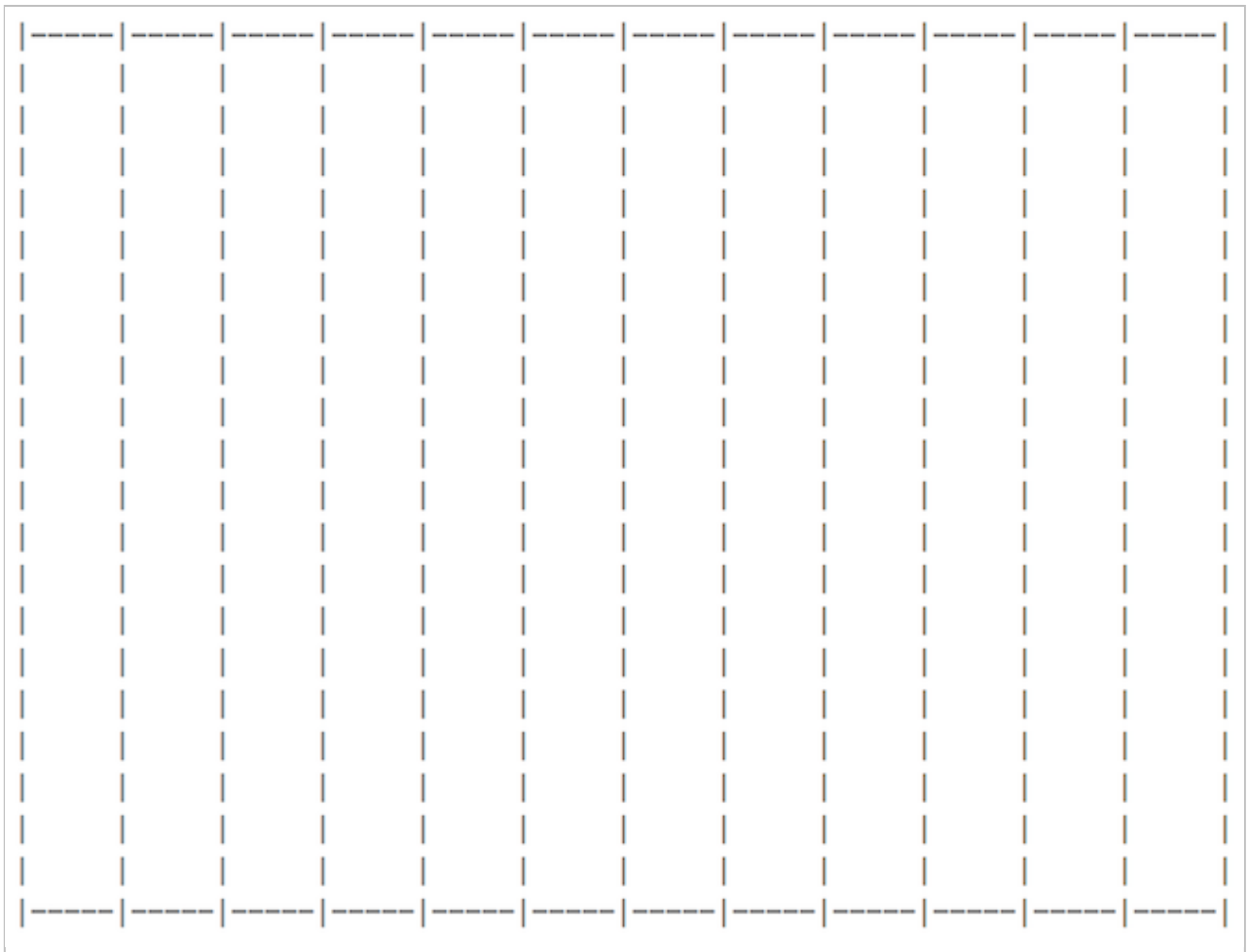
- alternative layouts
  - *grid*
  - *squares*
  - *snippet view*
  - *table*
  - *lists...*
- notifications
- snippets with expansion
- split views
  - *note snippet with contextual/media per note...*
- drag and drop delete
- filters
- sort options
- tags
- much more...

CSS grid layout - part 1

intro

- grid designs for page layout, components...
  - *increasingly popular over the last few years*
  - *useful for creating responsive designs*
- quick and easy to layout a scaffolding framework for our structured content
- create boxes for our content
  - *then position them within our grid layout*
- content can be stacked in a horizontal and vertical manner
  - *creating most efficient layout for needs of a given application*
- another benefit of CSS grids is that they are framework and project agnostic
  - *thereby enabling easy transfer from one to another*
- concept is based upon a set number of columns per page with a width of 100%
- columns will increase and decrease relative to the size of the browser window
- also set break points in our styles
  - *helps to customise a layout relative to screen sizes, devices, aspect ratios...*
  - *helps us differentiate between desktop and mobile viewers*

Image - Grid Layout



Grid Layout - Columns and rows

CSS grid layout - part 2

grid.css

- build a grid based upon 12 columns
  - *other options with fewer columns as well*
- tend to keep our grid CSS separate from the rest of the site
  - *maintain a CSS file just for the grid layout*
- helps abstract the layout from the remaining styles
  - *makes it easier to reuse the grid styles with another site or application*
- add a link to this new stylesheet in the `head` element of our pages

```
<link rel="stylesheet" type="text/css" href="assets/styles/grid.css">
```

- ensure padding and borders are included in total widths and heights for an element
  - *reset `box-sizing` property to include the `border-box`*
  - *resetting box model to ensure padding and borders are included*

```
* {
box-sizing: border-box;
}
```

CSS grid layout - example - part 3

grid.css

- set some widths for our columns, 12 in total
  - *each representing a proportion of the available width of a page*
  - *from a 12th to the full width of the page*

```
.col-1 {width: 8.33%;}
.col-2 {width: 16.66%;}
.col-3 {width: 25%;}
.col-4 {width: 33.33%;}
.col-5 {width: 41.66%;}
.col-6 {width: 50%;}
.col-7 {width: 58.33%;}
.col-8 {width: 66.66%;}
.col-9 {width: 75%;}
.col-10 {width: 83.33%;}
.col-11 {width: 91.66%;}
.col-12 {width: 100%;}
```

- classes allow us to set a column span for a given element
  - *from 1 to 12 in terms of the number of grid columns an element may span*

CSS grid layout - example - part 4

grid.css

- then set some further styling for each abstracted `col-` class

```
[class*="col-"] {
  position: relative;
  float:left;
  padding: 20px;
  border: 1px solid #333;
}
```

- create columns by wrapping our content elements into rows
- each row always needs 12 columns

```
<div class="row">
  <div class="col-6">left column</div>
  <div class="col-6">right column</div>
</div>
```

CSS grid layout - example - part 5

grid.css

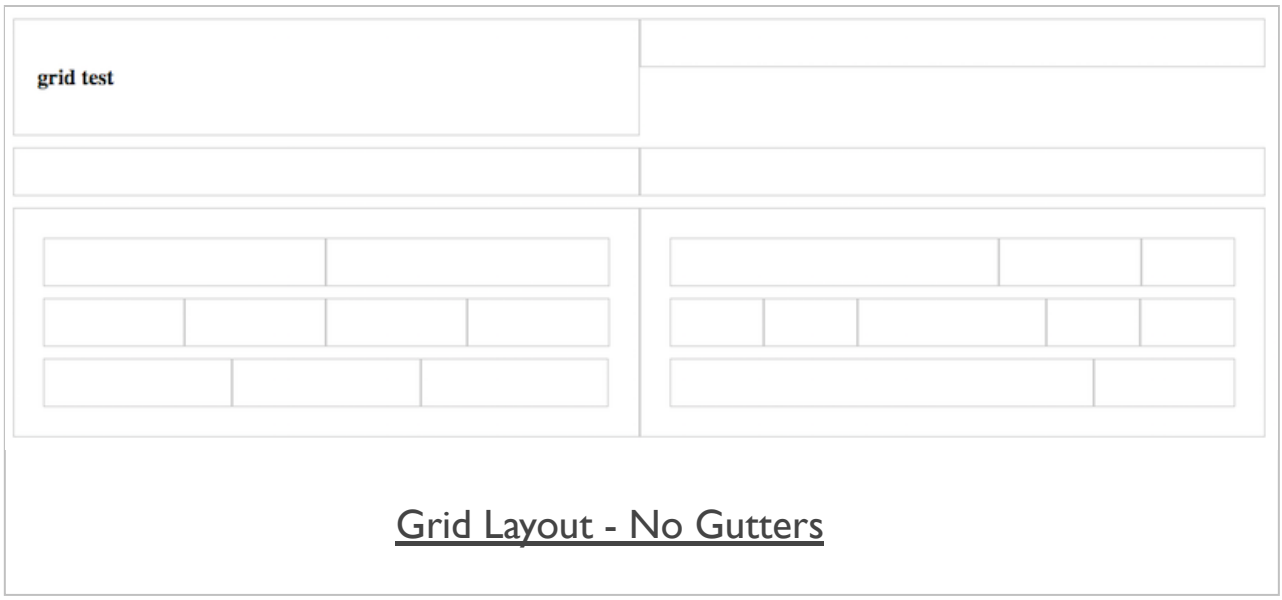
- due to the initial CSS of float left, each column is floated to the left
- columns are interpreted by subsequent elements in the hierarchy as non-existent
  - *initial placement will reflect this design*
- prevent this issue in layout, add the following CSS to grid stylesheet

```
.row:before, .row:after {
  content: "";
  clear: both;
  display: block;
}
```

- benefit of the clearfix, `clear: both`
  - *make row stretch to include columns it contains*
  - *without the need for additional markup*

DEMO - Grid Layout 1 - no gutters

Image - Grid Layout 1



Grid Layout - No Gutters

CSS grid layout - example - part 6

grid.css

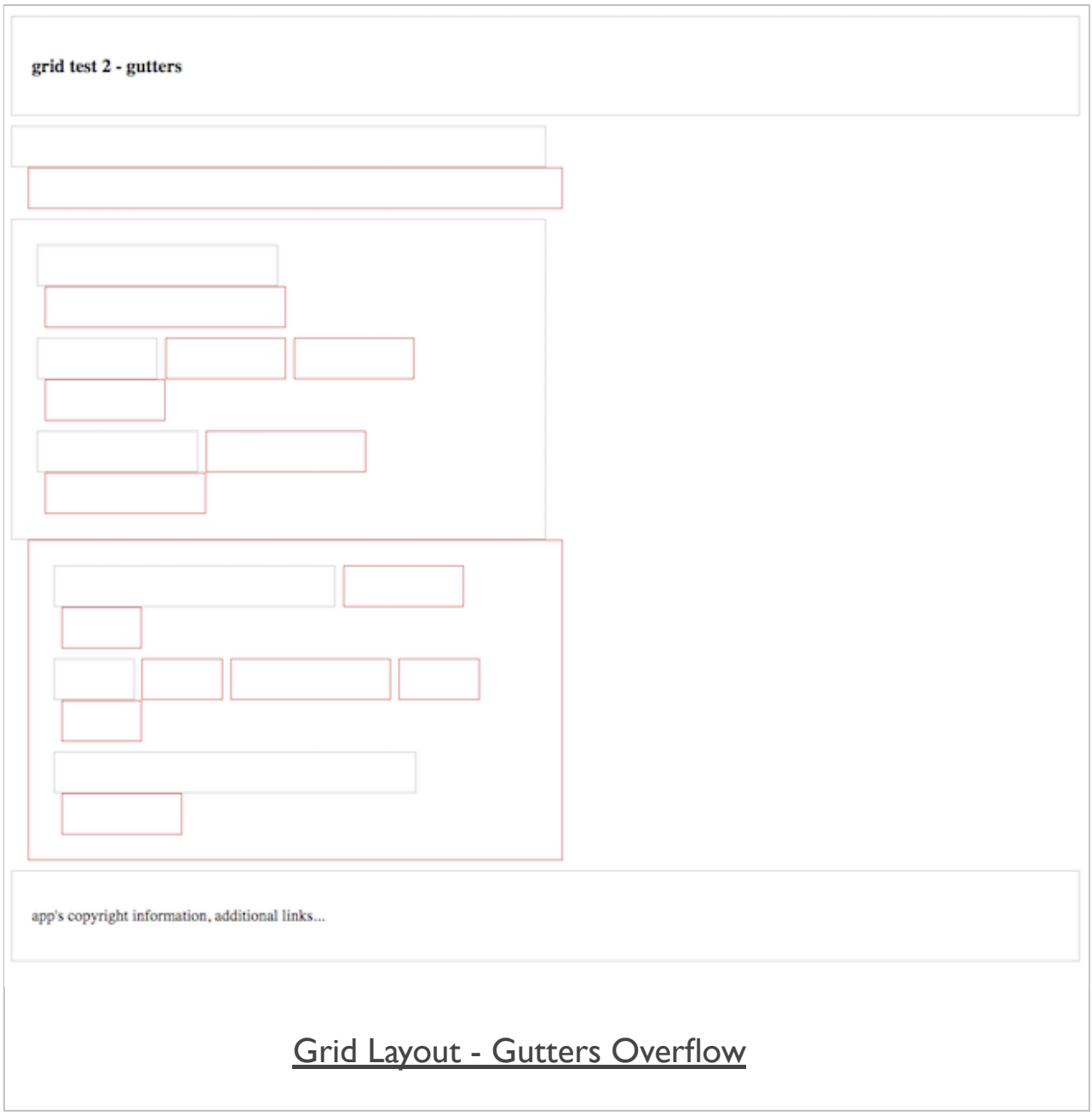
- add gutters to our grid to help create a sense of space and division in the content
- simplest way to add a gutter to the current grid css is to use padding
  - *rows can use padding, for example*

```
.row {
  padding: 5px;
}
```

- issue with simply adding padding to the columns
  - *margins are left in place, next to each other*
  - *column borders next to each with no external column gutter*
- fix this issue by targeting columns that are a sibling to a preceding column
- means we do not need to modify the first column, only subsequent siblings

```
[class*="col-"] + [class*="col-"] {
  margin-left: 1.6%;
}
```

Image - Grid Layout 2



Grid Layout - Gutters Overflow

CSS grid layout - part 7

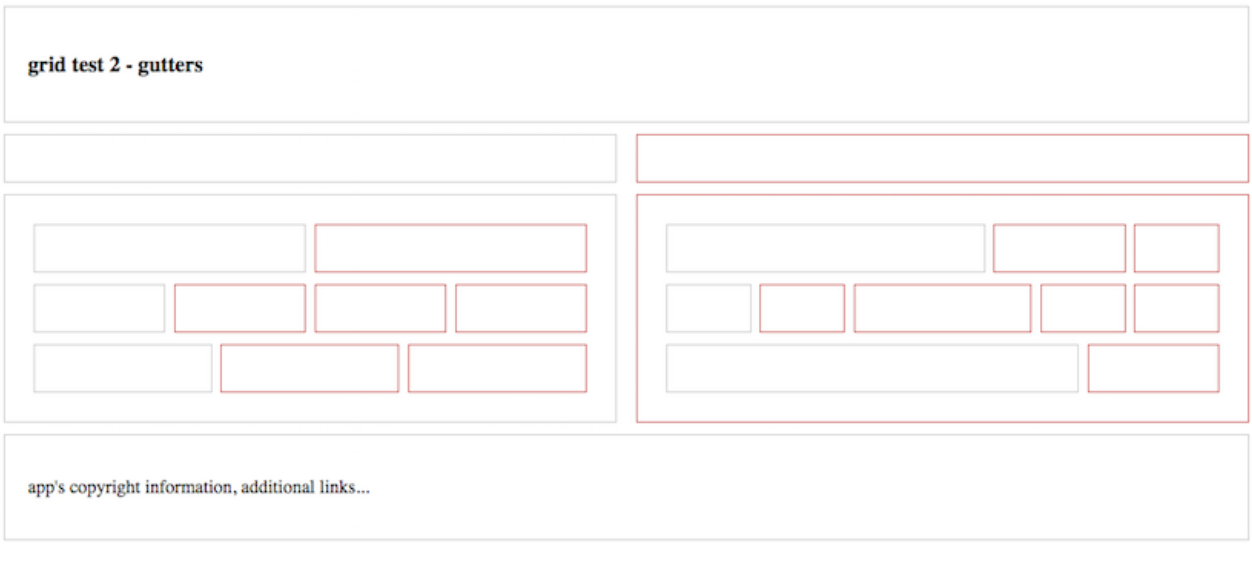
grid.css

- to fix this issue we recalculate permitted % widths for our columns in the CSS
  - *we now have % widths as follows*

```
.col-1 {width: 6.86%;}
.col-2 {width: 15.33%;}
.col-3 {width: 23.8%;}
.col-4 {width: 32.26%;}
.col-5 {width: 40.73%;}
.col-6 {width: 49.2%;}
.col-7 {width: 57.66%;}
.col-8 {width: 66.13%;}
.col-9 {width: 74.6%;}
.col-10 {width: 83.06%;}
.col-11 {width: 91.53%;}
.col-12 {width: 100%;}
```

DEMO - Grid Layout 2 - gutters

Image - Grid Layout 3



Grid Layout - Gutters

CSS grid layout - part 8

media queries

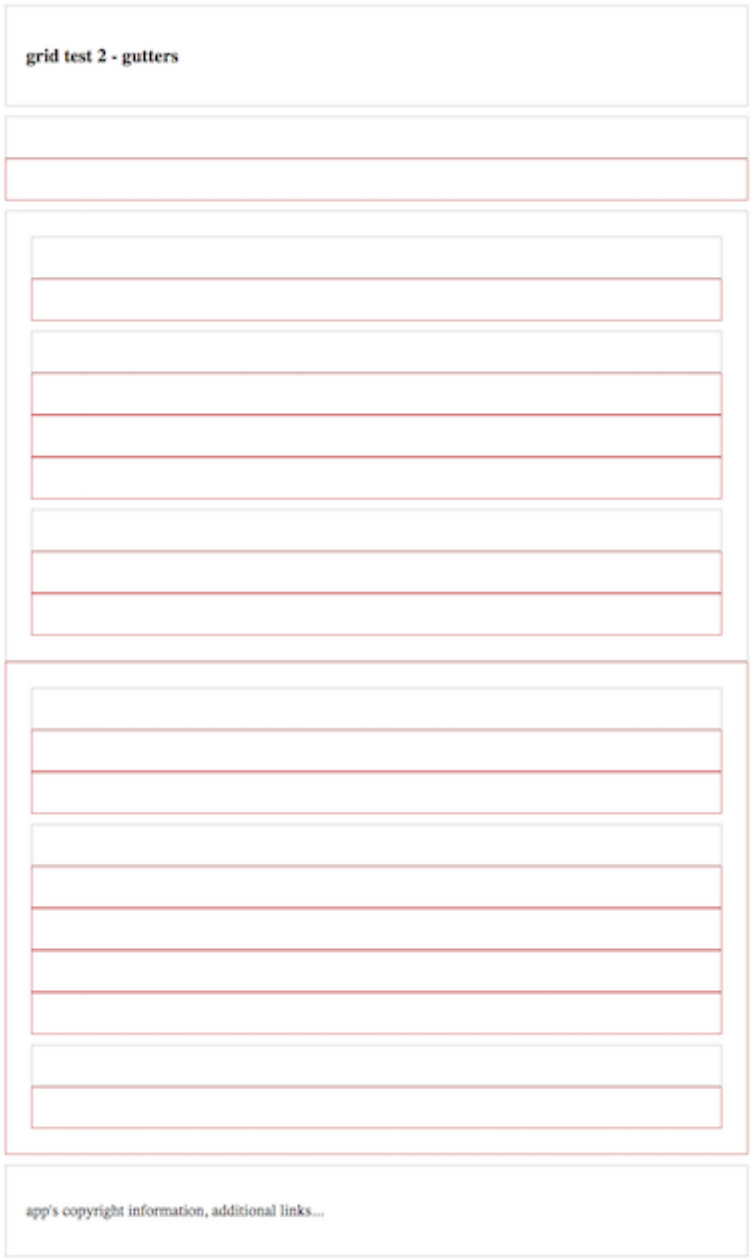
- often need to consider a mobile-first approach
- introduction of CSS3, we can now add **media queries**
- modify specified rulesets relative to a given condition
  - *eg: screen size for a desktop, tablet, and phone device*
- media queries allow us to specify a breakpoint in the width of the viewport
  - *will then trigger a different style for our application*
- could be a simple change in styles
  - *such as colour, font etc*
- could be a modification in the grid layout
  - *effective widths for our columns per screen size etc...*

```
@media only screen and (max-width: 900px) {
  [class*="col-"] {
  width: 100%;
  }
}
```

- gutters need to be removed
  - *specifying widths of 100% for our columns*

```
[class*="col-"] + [class*="col-"] {
  margin-left:0;
}
```

Image - Grid Layout 4



Grid Layout - Media Queries

HTML5, CSS, & JS - example - part 1

add grid layout

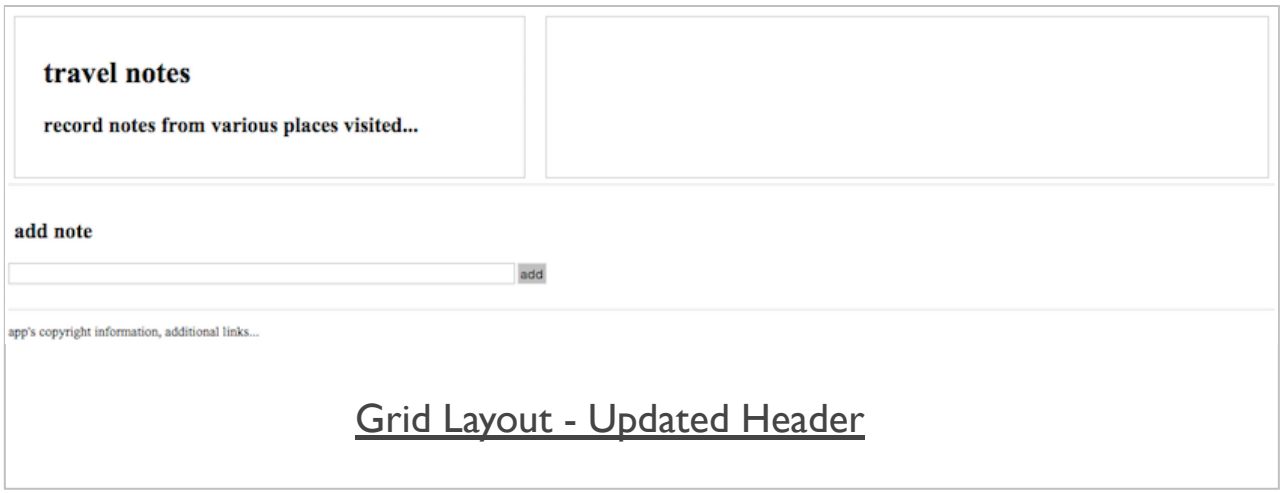
- update the layout of our Travel Notes application to include a grid layout
- apply this grid layout to the overall application
  - *organisation and presentation of the notes*
- remove the centred, fixed width for the `body` in our style.css stylesheet
- removes centre styling, results in content spanning full width of browser window
- add the grid layout to help us control this layout

```
<link rel="stylesheet" type="text/css" href="assets/styles/grid.css">
```

- then modify content categories, child elements to use new grid css

```
<!-- document header -->
<header>
  <div class="row">
    <div class="col-5">
      <h3>travel notes</h3>
      <h5>record notes from various places visited...</h5>
    </div>
    <div class="col-7"></div>
  </div>
</header>
```

Image - HTML5, CSS, & JS - grid layout



Grid Layout - Updated Header

HTML5, CSS, & JS - example - part 2

add grid layout

- update our main content to position the `note-input` and `note-controls`

```
<!-- note input -->
<section class="note-input">
  <div class="row">
    <div class="col-12">
      <h5>add note</h5>
      <input><button>add</button>
    </div>
  <div>
</section>
<!-- note controls for delete... -->
<section class="note-controls">
  <div class="row">
    <div class="col-12">
      <button id="notes-delete">Delete all</button>
    </div>
  </div>
</section>
```

- still need to amend style.css to remove additional fixed styling

Image - HTML5, CSS, & JS - grid layout 2

Grid Layout - mixed grid and fixed

HTML5, CSS, & JS - example - part 3

add grid layout

- fix mixed rendering by removing width, margin, and padding for `.note-controls`

```
/* note controls */
.note-controls {
  border-bottom: 1px solid #dedede;
  display: none;
}
```

- continue to update Travel Notes app
  - *modify output for notes*
  - *add further options for users*

DEMO - Travel Notes - grid layout with media queries

HTML5, CSS, & JS - example - part 4

add flex to grid layout

- an additional option to consider - flex layouts
  - *a recent W3 working draft*
  - *aims to provide efficient way to align and proportion content*
- known as **Flexbox Layout**
  - *idea is to apportion width and height for content*
  - *proportions relative to container even when their size is unknown or dynamic*
- flex layout could, in theory, replace a full grid layout
  - *considered more a complement to overall grid structure*
- defined flex container expands items to fill the container's available space
  - *can also shrink them to prevent any possible overflow*
- think of a flex layout as supporting multiple directions
  - *direction agnostic*
- many properties available for **flex**
  - *focus upon styling flex container and any flex items*

HTML5, CSS, & JS - example - part 5

add flex to grid layout

- we might specify CSS properties for a flex container

```
.flex-container {
display: flex; /* defines container as flex */
flex-direction: row; /* defines positioning of items added to container */
flex-wrap: wrap; /* defines whether to wrap items to another line */
justify-content: flex-start; /* defines start point and distribution of items */
}
```

- allows us to position our container starting at the left
  - *items contained in a row*
  - *contained items wrapping to additional lines if necessary*
- many additional options available for each property
- also add rulesets for specific styling of items within a flex container
- we could add properties to a flex item such as
  - *specify the order of the flex items*
  - *whether a particular item can grow or shrink relative to content*
  - *default size of an item before any remaining space is distributed*
  - *individual alignment for a given item...*

HTML5, CSS, & JS - example - part 6

add flex to notes

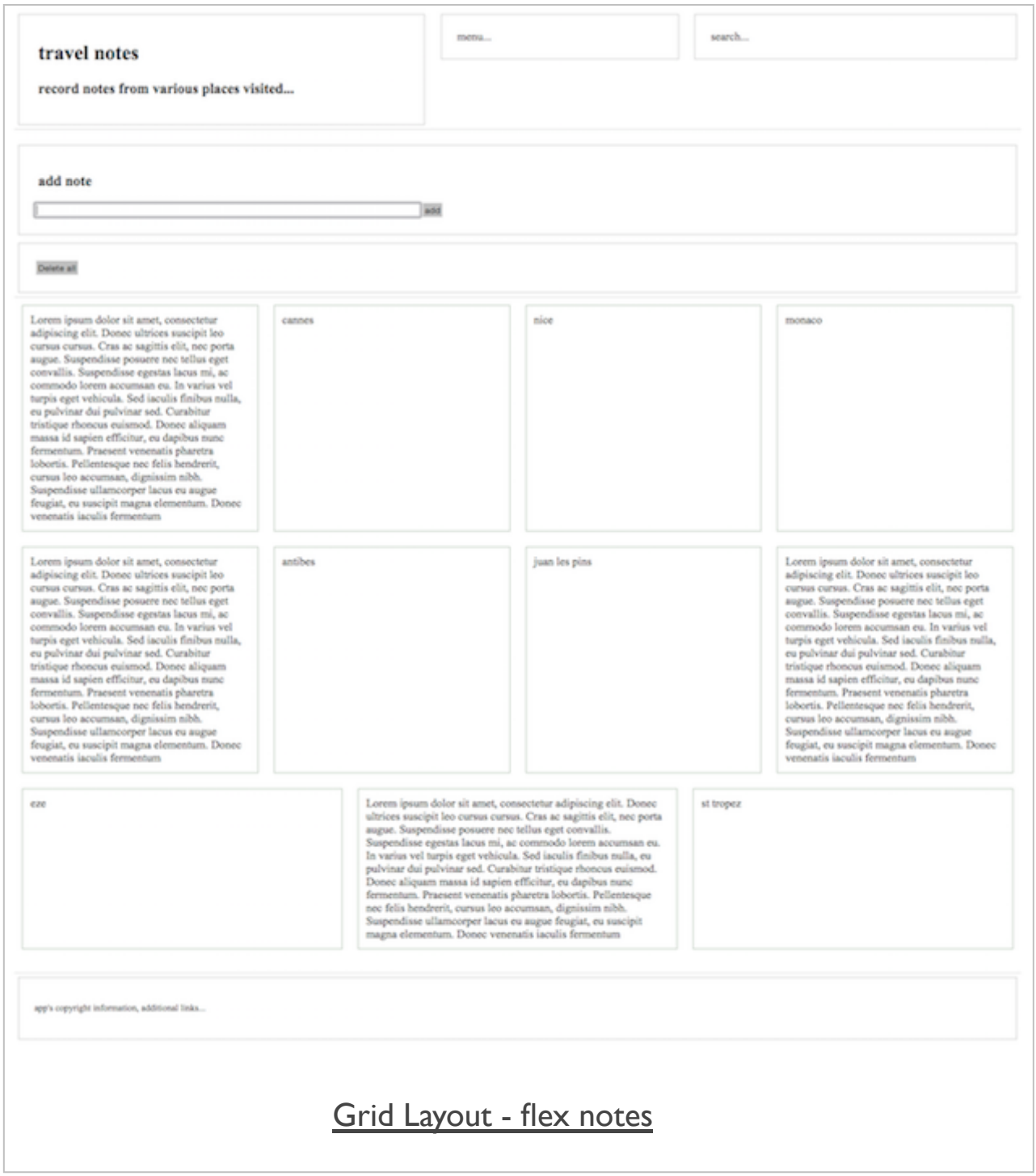
- flex container and items useful for organising and positioning our notes
- due to uncertainty about content, size, and general note requirements
  - *flex positioning and styling removes the need for assumptions or fixed sizes*
- we can start to modify the styling and rendering of our notes using flex

```
/* flex item */
.flex-item {
  flex-basis: 300px; /* default size before extra */
  flex-grow: 1; /* all items will be equal */
}
```

- gives us a default smallest size for each note
- then the ability for each note to grow to fill the row as required
- also work with responsive layouts
  - *due to the minimum size and the option to grow for each item*
  - *and wrap flex items per flex container*
- modify and update styles as we develop travel notes app

DEMO - Travel Notes - grid layout with flex notes

Image - HTML5, CSS, & JS - Flex Notes

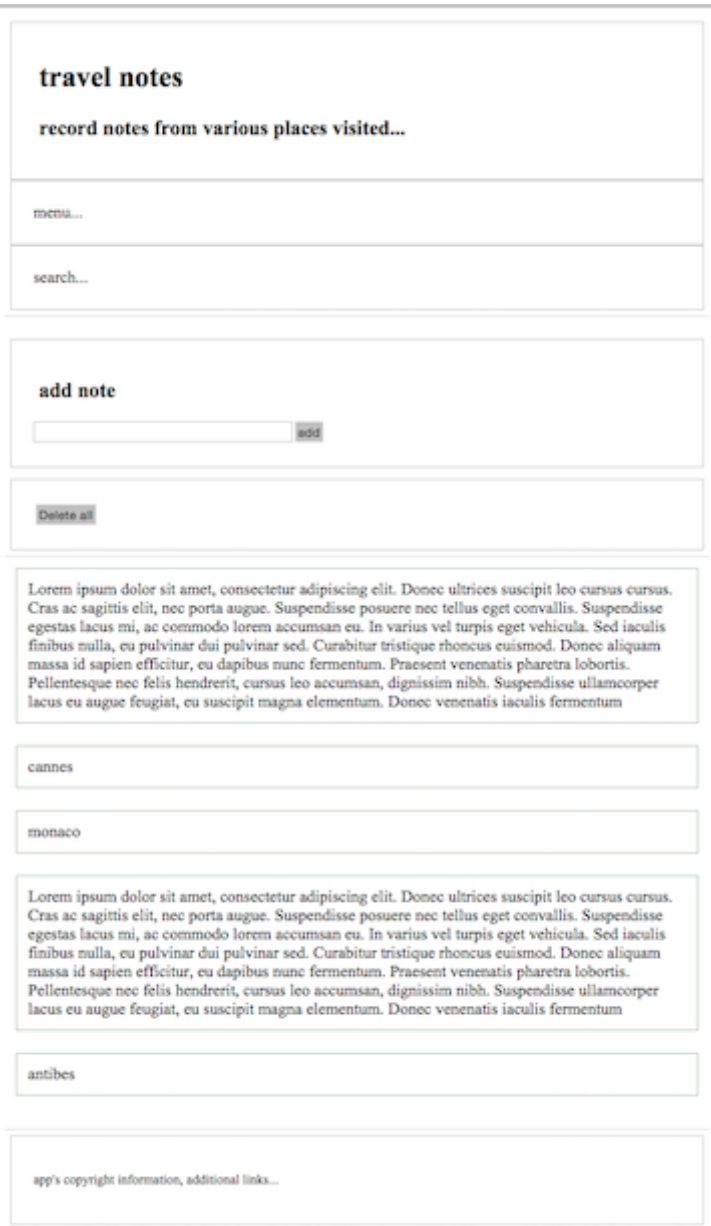


Grid Layout - flex notes

Image - HTML5, CSS, & JS - Flex Notes 2

Grid Layout - flex notes - medium

Image - HTML5, CSS, & JS - Flex Notes 3



Grid Layout - flex notes - small

HTML5, CSS, & JS - example - part 7

add flex to notes

Notes with Flex and Media Queries

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 explicit deferred object
  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()

HTML5, CSS, & JS - example - part 1

add AJAX and JSON - load notes from json

- update our **travel notes** app to allow us to load some test persistent notes from a local JSON file
- initial JSON is as follows

```
{
  "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..."
  }]
}
```

HTML5, CSS, & JS - example - part 2

add AJAX and JSON - load notes from json

- add option to load notes from JSON as app initially loads
  - *use deferred promise pattern*
  - *checks source JSON as it loads via the promise*
  - *then outputs the end result*
- start with the following update

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

HTML5, CSS, & JS - example - part 3

add AJAX and JSON - load notes from json

- help us better manage logic of our notes from app's loading and execution
  - *create two separate JS files*
- our updated structure might be as follows

```
...
|- assets
   |- scripts
      |- travel.js
      |- notes.js
...
```

- we can extend this further, as needed by app features and data

HTML5, CSS, & JS - example - part 4

add AJAX and JSON - load notes from json

- add our `.when()` function to the app's loader
  - *`.when()` function accepts a deferred object*
  - *in our case a limited promise*
- then allows us to chain additional deferred functions
  - *including required `.done()` function*
- for returned data, use standard response object to get `travelNotes`
  - *then iterate over the array for each property*
  - *for each iteration, we can simply call our `createNote` function*
  - *builds and renders required notes to the app's DOM*

```
//use deferred object from getJson
$.when(getNotes()).done(function(response) {
  //get travelNotes object
  var $travelNotes = response.travelNotes
  //process travelNotes array
  $travelNotes.forEach(function(item) {
    //check each property
    if (item !== null) {
      //get note
      var note = item.note;
      //create each note for rendering
      createNote(note);
    }
  });//end foreach
});
```

HTML5, CSS, & JS - example - part 5

add AJAX and JSON - load notes from json

- simple problem - existing `createNote()` function does not accept a parameter
- need to update the logic of that function to accept and handle a parameter
- also requires a quick update to any functions and calls to the `createNote()`
  - *event handlers for creating a new note using the `add` button and keypress within the input field*

```
//manage input field and new note output
function createNote(data) {
  ...
  //conditional check for data
  if (data !== "") {
  //set content for note
  $note.html(data);
  ...
  }
}
```

HTML5, CSS, & JS - example - part 6

add AJAX and JSON - load notes from json

- update our event handlers for the note input button and input field keypress as follows,

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

```
//handle user event for keyboard press
$(".note-input input").on("keypress", function(e) {
  //check code for keyboard press
  if (e.keyCode === 13) {
    var $note_data = getNoteInput();
    //call note builder function
    createNote($note_data);
  }
});
```

- our notes now load by default as the app starts
- note input button and keypress work as expected
- DEMO - travel notes & JSON

Working with APIs - part 1

remote api options - Flickr

- **Travel Notes** app loads data from a local JSON file
- add option to load different types of data using remote APIs
  - *Flickr API for images, tags...*
- basics and principles are similar to the patterns we've already seen and tested
- test a sample JSON return from the Flickr API
- JSON return - useful properties for app
  - *title*
  - *link*
  - *media (direct url for image - where available)*
  - *description*
  - ...
- public feed for searching public photos, videos, groups, recent activity...
- Flickr API Public Feed - Cannes and France

Working with APIs - part 2

working with Flickr API

- query Flickr's public feed for photos
  - *we can use our now familiar pattern for requesting JSON*

```
//get the Flickr public feed JSON for images
function getImages() {
  //.get returns an object derived from a Deferred object - do not need explicit deferred object
  var $deferredNotesRequest = $.getJSON (
  "http://api.flickr.com/services/feeds/photos_public.gne?jsoncallback=?",
    { tags: "cannes,france,boules",
      tagmode: "all",
      format: "json"
    });
  return $deferredNotesRequest;
}
```

- need to make a few specific modifications to the request
  - *JSONP to avoid browser security restrictions*

Working with APIs - part 3

working with Flickr API

- Flickr's public feed includes options
  - *eg: a specific user ID for photos, various tags, how tags are interpreted by the search...*
- use our `.when()` function to load and render some test images from Flickr

```
$.when(getImages()).done(function(response) {
  console.log("done..."+response);
  //use jQuery's generic iterative function for the response...
  $.each( response.items, function( i, item ) {
    buildImage(item.media.m);
    //limit test images to 8
    if ( i === 7 ) {
      return false;
    }
  });
});
```

- DEMO - AJAX and JSON - Flickr api

Demos

AJAX

- DEMO 1 - AJAX - demo 1
- DEMO 2 - AJAX - demo 2

AJAX and JSON

- AJAX-JSON 1 - load a JSON file
- AJAX-JSON 2 - abstract code for load a JSON file
- AJAX-JSON 3 - test deferred .then()
- AJAX-JSON 4 - Flickr API

Grids

- Grids 1 - Grid 1 with no gutters
- Grids 2 - Grid 2 with gutters

Travel notes app - series 3

- DEMO 1 - Travel notes - grid layout with media queries
- DEMO 2 - Travel notes - demo2

Travel notes app - series 4

- DEMO 1 - Travel Notes & JSON

References - JS & Libraries

- Flickr API
- Public feeds
- Public feed - public photos & video
- jQuery
- jQuery
- jQuery API
- jQuery - deferred
- jQuery - .getJSON()
- jQuery - JSONP
- jQuery :parent selector
- jQuery - promise
- MDN
- MDN - JS Objects
- W3
- W3 - CSS Flexible Box Layout Module 1