Comp 324/424 - Client-side Web Design

Fall Semester 2018 - Week 13

Dr Nick Hayward

ES2017 Async & Await

- in ES2017, JavaScript gained native syntax to describe asynchronous operations
- now use *async/await* to work with asynchronous operations
- `Async` functions allow developers to take a promise-based implementation
  - *then use synchronous-like patterns of a generator*
  - *e.g. async implementation with sync usage patterns...*
- `await` may only be used inside `async` functions
  - *denoted with the `async` keyword*
- `async` function works in a similar manner to standard generators
  - *e.g. suspending execution in local context until a promise settles*
- if awaited expression is not originally a promise object
  - *it will be cast to a promise in this context...*

ES2017 Async & Await - example 1

- example usage with try/catch

```
async function read() {
  // use try/catch to handle errors in awaited promises within async function
  try {
    const model = await getRandomBook();
  } catch (err) {
    console.log(err);
  }
}
// call function as usual
read();
```

- use return Promise object

```
async function read() {
  const model = await getRandomBook();
}
// call function as usual - work with return promise object...
read()
  .then()
```

ES2017 Async & Await - example 2

```
/*
* basic-async1.js
* async called with sync-like try/catch block
* 'awaits' return from fetch to local JSON file
*/

// FN: 'fetch' from JSON
function getNotes() {
  return fetch('./assets/files/notes.json', {
    headers: new Headers({
      Accept: 'application/json'
    })
  })
  .then(res => res.json());
}

// FN: async/await
async function read() {
  try {
    const notes = await getNotes();
    console.log(`notes FETCH successful`);
  } catch (err) {
    console.log(err);
  }
}

read();
```

- Demo - Async & Await - Fetch example

ES2017 Async & Await - example 3

initial `fetch`

```
// FN: 'fetch' from JSON
function getNotes() {
  return fetch('./assets/files/notes.json', {
    headers: new Headers({
      Accept: 'application/json'
    })
  })
  .then(res => res.json());
}
```

ES2017 Async & Await - example 4

iterable functions

```
/*
* FNs: iterable computed data
* functions support all major ES6 data structures
* - arrays, typed arrays, maps, sets...
*/

// FN: iterable entries() - default iterator for data structure entries
function dataEntryIterator(data) {
  for (const pair of data.entries()) {
    console.log(pair);
  }
}

// FN: iterable keys() - default iterator for data structure keys
function dataKeysIterator(data) {
  for (const key of data.keys()) {
    console.log(key);
  }
}

// FN: iterable values() - default iterator for data structure values
function dataValuesIterator(data) {
  for (const value of data.values()) {
    console.log(value);
  }
}
```

ES2017 Async & Await - example 5

async and await usage - a bit of fun...

```
// FN: async/await
async function read() {
  try {
    // await return from FETCH for notes.json file
    const data = await getNotes();
    const notes = data['notes'];
    // wrap return notes array in iterator
    const iter = notes[Symbol.iterator]();
    // test iterator with next for each result...
    console.log(iter.next());
    console.log(iter.next());
    console.log(iter.next());
    console.log(iter.next());
    console.log(`notes FETCH successful`);
    dataEntryIterator(notes);
    dataKeysIterator(notes);
    dataValuesIterator(notes);
  } catch (err) {
    console.log(err);
  }
}

read();
```

- Demo - Async & Await - example with iterables

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

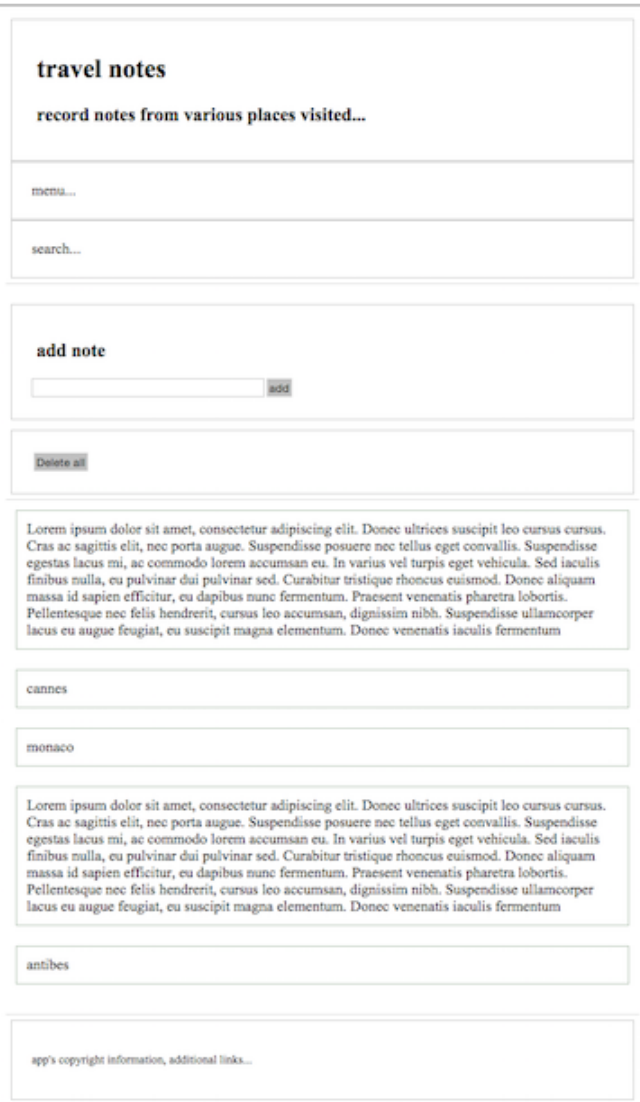
Image - HTML5, CSS, & JS - Flex Notes

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

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
  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
  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

HTML5, CSS, & JS - example - part 7

working with Flickr API - update travel notes

- add option to Travel Notes app to allow a user to view images from Flickr
- need to update app's HTML, CSS, and JS
- modify how our notes, and associated options, are rendered to our users
- add a search option for photos on Flickr
- render our images to match the notes
- app's structure still reflects three primary content categories
  - *`header`, `main`, and `footer` with slight modifications to the `main` category*
- `main` content category updated to create two distinct rows for initial content
  - *contain defined semantic containers*
- row containing `.note-input` and Flickr search option `.contextual-choice`
  - *then split this row into two columns of 6*

HTML5, CSS, & JS - example - part 8

working with Flickr API - update travel notes HTML

- updated HTML for `.note-input` and Flickr search `.contextual-choice`

```
<div class="row">
  <!-- note input -->
  <section class="note-input col-6">
    <h5>add note</h5>
    <input><button>add</button>
  </section>
  <!-- contextual choice -->
  <section class="contextual-choice col-6">
    <h5>search flickr</h5>
    <input><button>search</button>
  </section>
</div>
```

HTML5, CSS, & JS - example - part 9

working with Flickr API - update travel notes HTML

- update the HTML for rendering the images
  - *add alongside our notes*
- create another row for these containers
  - *add two section containers for `.note-output` and `.contextual-output`*
- make `.note-output` slightly larger to show primary app focus

```
<div class="row">
  <!-- note output -->
  <section class="note-output col-7 flex-container">
  </section>
  <!-- contextual output -->
  <section class="contextual-output col-5 flex-container">
  </section>
</div>
```

HTML5, CSS, & JS - example - part 10

working with Flickr API - update travel notes JS

- add further functionality to **Travel Notes** app
- split our JS logic into three files to help with oranisation
  - *a main loader file, `travel.js`,*
  - *and a file each for notes and contextual options*
- updated app structure for JS

```
...

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

- underlying logic for the notes will remain the same
  - *move loading of default notes to the `travel.js` main loader file*
- updates for searching, returning, and rendering images from Flickr
  - *added to the `contextual.js` file*

HTML5, CSS, & JS - example - part 11

working with Flickr API - update travel notes JS

- test Flickr API in our app using some set data for image tags
  - *respond to the user clicking on the search button*
  - *submit our query to Flickr*
  - *process the returned JSON for the images*
  - *render them for viewing*
- request and process our images using the familiar pattern

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

HTML5, CSS, & JS - example - part 12

working with Flickr API - update travel notes JS

- returned data using standard deferred promise object
  - *add a new function to handle the processing of the images*

```
function processImages(data) {
  $.when(getImages($img_data)).done(function(response) {
    //use jQuery's generic iterative function for the response...
    $.each( response.items, function( i, item ) {
      createImage(item.media.m);
      //limit test images to 4
      if ( i === 3 ) {
        return false;
      }
    });
  });
}
```

- using deferred promise object with `.when()` function chained to `.done()` function
- add jQuery's generic iterative function to help us process the response
  - *instead of standard JavaScript `.forEach()` option*
- loop through each value, and pass the image to our new function, `createImage()`
  - *ready for rendering to our app's DOM*
  - *limit number of images for testing*

HTML5, CSS, & JS - example - part 13

working with Flickr API - update travel notes JS

```
//manage new image output
function createImage(data) {
  //create each image element
  var img = $('<img class="flex-img">');
  //add image
  img.attr("src", data);
  //append to DOM
  $(".contextual-output").append(img);
}
```

- `.createImage()` function accepts a parameter for image data
- then process ready for rendering to the app's DOM
- image is added to a new `img` element with a new class of `.flex-img`
  - *creates a flex item for rendering*
- added to the new `.contextual-output` section
- rendered images displayed as thumbnails for the user
  - *complementary to the existing notes*

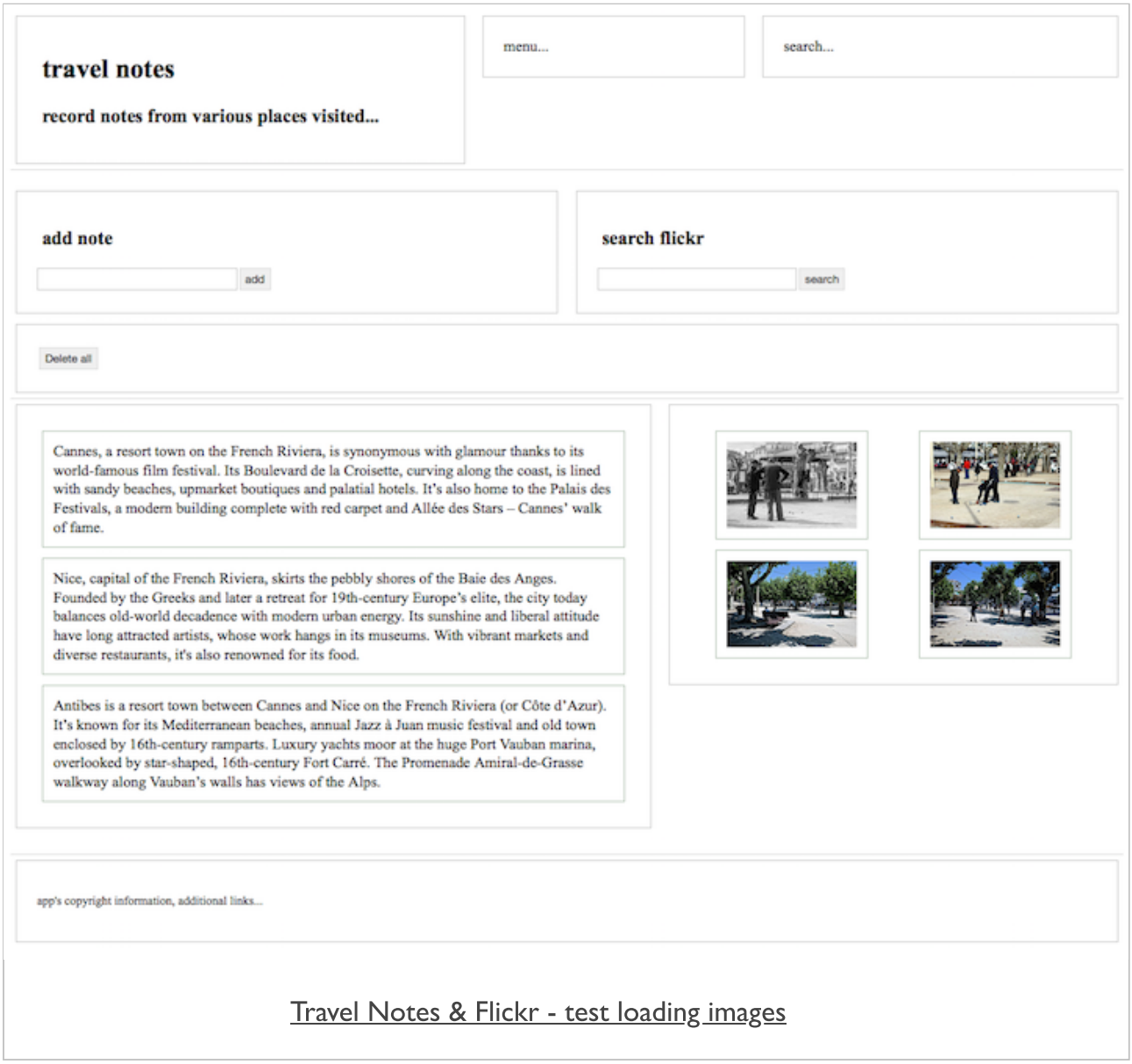
HTML5, CSS, & JS - example - part 14

working with Flickr API - update travel notes JS

- to add images to the app
  - *a user can enter their requested tags in the search field*
  - *then click on the `search` button to return any available images*
- event handler for this `search` button click uses the requested tags
  - *passes them as a parameter to the `processImages()` function*

```
//handle user event for image `search` button click
$(".contextual-choice button").on("click", function(e) {
  //test tags for testing image search
  $img_data = "cannes,france,boules"
  //process images
  processImages($img_data);
});
```

Image - HTML5, CSS, & JS - Travel Notes & Flickr



Travel Notes & Flickr - test loading images

HTML5, CSS, & JS - example - part 15

working with Flickr API - update travel notes CSS

- need to update and modify existing CSS
  - *helps with correct rendering of the thumbnail images*
- CSS additions are initially modest
  - *reflects integration with existing app, grid, and flex layouts*
- add new ruleset for image rendering in the `.contextual-output` section

```
/* contextual output images */
.contextual-output img {
  margin: 5px;
  padding: 5px;
  border: 1px solid #b1c4b1;
}
```

- update `.flex-container` class to change `justify-content` property to value of `space-around`
- add new ruleset for a `.flex-img` class.

```
/* flex image */
.flex-img {
  flex-basis: 150px;
  flex-grow:0;
}
```

- specify size of a thumbnail image
  - *initially restrict their ability to grow relative to flex*

HTML5, CSS, & JS - example - part 16

working with Flickr API - update travel notes JS

- we can now request, process, and render images from Flickr to Travel Notes app
  - *still need to accept and process search queries from search input field.*
- add option to check search input field
  - *then submit query to Flickr for images*

```
//get input value for image search
function getImageInput() {
  //define img value
  var img_val = "";
  //define input field
  var $img_tags = $(".contextual-choice input");
  if ($img_tags.val() !== "") {
    img_val = $img_tags.val();
    return img_val;
  } else {
    return img_val;
  }
}
```

HTML5, CSS, & JS - example - part 17

working with Flickr API - update travel notes JS

- use `getImageInput()` function with a modified `processImages()` function

```
//process image production, loading, and pass to rendering
function processImages() {
  //check img visibility for contextual-output - clear existing images
  if (checkVisible($(".contextual-output img")) === false) {
    //empty existing images
    $(".contextual-output").empty();
  }
  //get data from image search input field
  var $img_data = getImageInput();
  //use image data to get images, and pass for rendering
  $.when(getImages($img_data)).done(function(response) {
    console.log("done..."+response);
    //use jQuery's generic iterative function for the response...
    $.each( response.items, function( i, item ) {
      createImage(item.media.m);
      //limit test images to 4
      if ( i === 3 ) {
        return false;
      }
    });
  });
}
```

HTML5, CSS, & JS - example - part 18

working with Flickr API - update travel notes JS

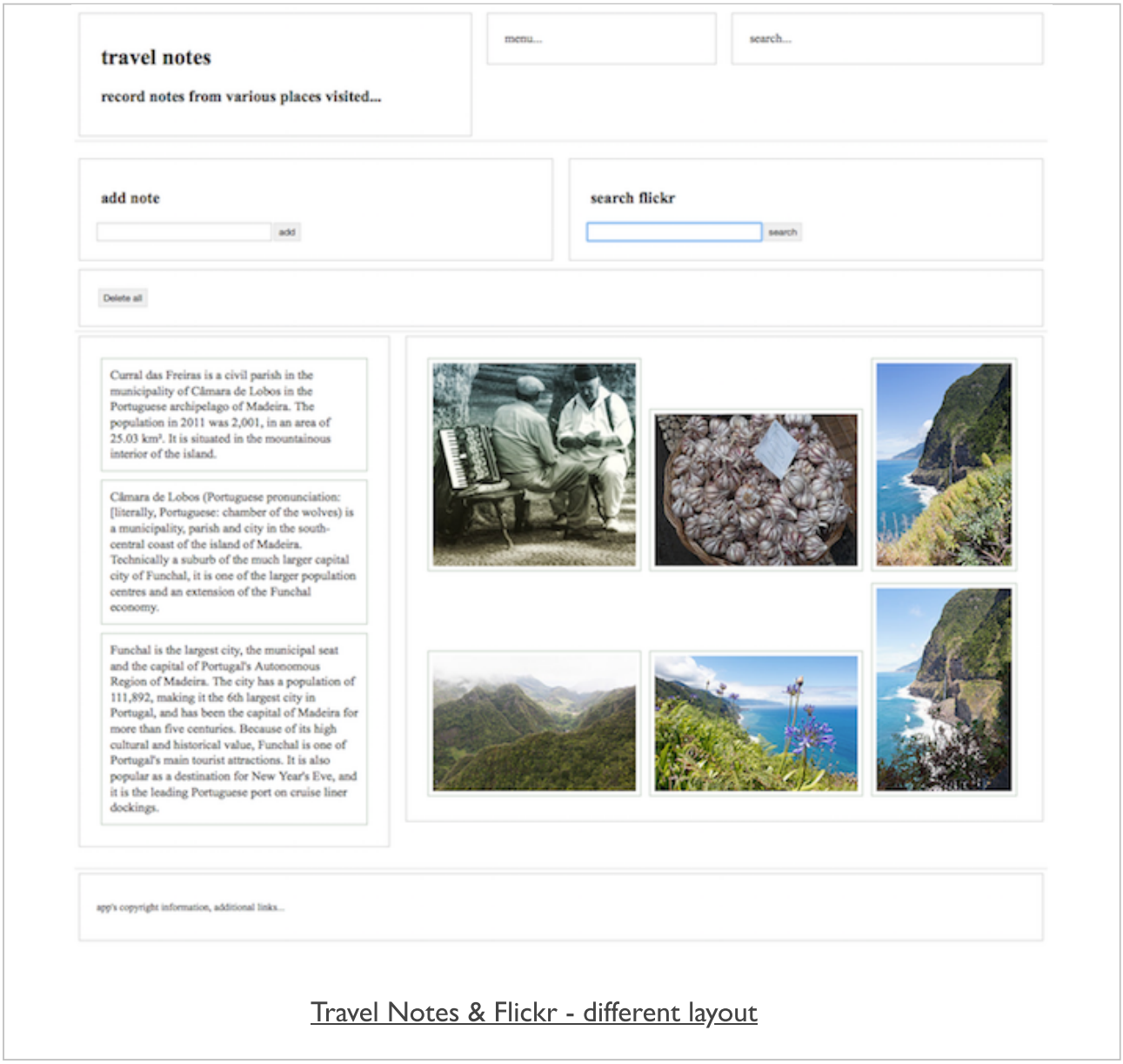
- updated `processImages()` function then called within event handlers
  - *for the search button and a keypress in the search input field*

```
//handle user event for image search button click
$(".contextual-choice button").on("click", function(e) {
  //process images
  processImages();
});

//handle user event for keyboard press
$(".contextual-choice input").on("keypress", function(e) {
  //check code for keyboard press
  if (e.keyCode === 13) {
    //process images
    processImages();
  }
});
```

- DEMO - travel notes & Flickr

Image - HTML5, CSS, & JS - Travel Notes & Flickr



Travel Notes & Flickr - different layout

HTML5, CSS, & JS - example - part 19

working with Flickr API - update travel notes JS

- room for improvement, updates, abstraction, and general refactoring of the existing code
- return to this issue when we consider refactoring the code in general
  - *there are still a few simple features we need to add*
- for example,
  - *add images to the* `.contextual-output` *section, resize* `.note-output` *section*
  - *moves focus to the current images*
  - *check loading progress of the notes and images*
  - *show feedback to the user*
  - *need to output a title for the images*
  - *set using the search query*

HTML5, CSS, & JS - example - part 20

working with Flickr API - modify travel notes JS

- first modification is to resize the `.notes-output`
  - *create more space for the images*
  - *gently shift focus to the new images*
- update existing `.createImage()` function in the `contextual.js` file

```
//manage new image output
function createImage(data) {
...
    if (checkVisible($(".contextual-output img")) === true) {
      $(".note-output").removeClass("col-12");
      $(".note-output").addClass("col-4");
      $(".contextual-output").fadeIn("slow");
    }
...
}
```

- add check to ensure images are not visible in the DOM
- remove current class from `.note-output` section
  - *12 column class for the grid*
- add new grid class to resize `.note-output` to 4 columns
  - *then fade in the `.contextual-output` class*
  - *set in the app's HTML to a class of `.col-8`*

HTML5, CSS, & JS - example - part 21

working with Flickr API - modify travel notes JS

- next modification is some initial error handling
  - *checking for an empty array of images from the returned Flickr JSON*
- check `processImages()` function for an empty array of image items

```
...

if (response.items.length === 0) {
  var img = "";
  createImage(img);
} else {
  //return images from items array...
}
...
```

- checks images in the items array for the promise object
- if not, send an empty variable as a parameter to our `createImage()` function

HTML5, CSS, & JS - example - part 22

working with Flickr API - modify travel notes JS

- check for empty value in `createImage()` function
  - *handle the simple errors as follows*

```
if (data !== "") {
  //create each image element
  var $img = $('<img class="flex-img">').attr("src", data);
  //add image
  img_output = $img;
} else {
  var $img_error = $('<p class="flex-item error">').html("No images available...");
  //add error
  img_output = $img_error;
}
```

- we've abstracted the return variable for the image output
  - *can hold either the image or the error output...*
- add a check to see whether the `.contextual-output` section is visible or not
- modify the column class for the `.note-output` section
- then append our image output
- then show the `.contextual-output` section within the app
- DEMO - travel notes & Flickr

Image - HTML5, CSS, & JS - Travel Notes & Flickr

Travel Notes & Flickr - error checking

HTML5, CSS, & JS - example - part 23

working with Flickr API - modify travel notes JS

- continue to modify and build our Travel Notes app
- add some metadata for the returned images
  - *using the title and link from the search query response*
- add initial metadata output in the `contextual.js` file
  - *modify the `processImages()` function*
  - *metadata from Flickr JSON response in the deferred promise object*

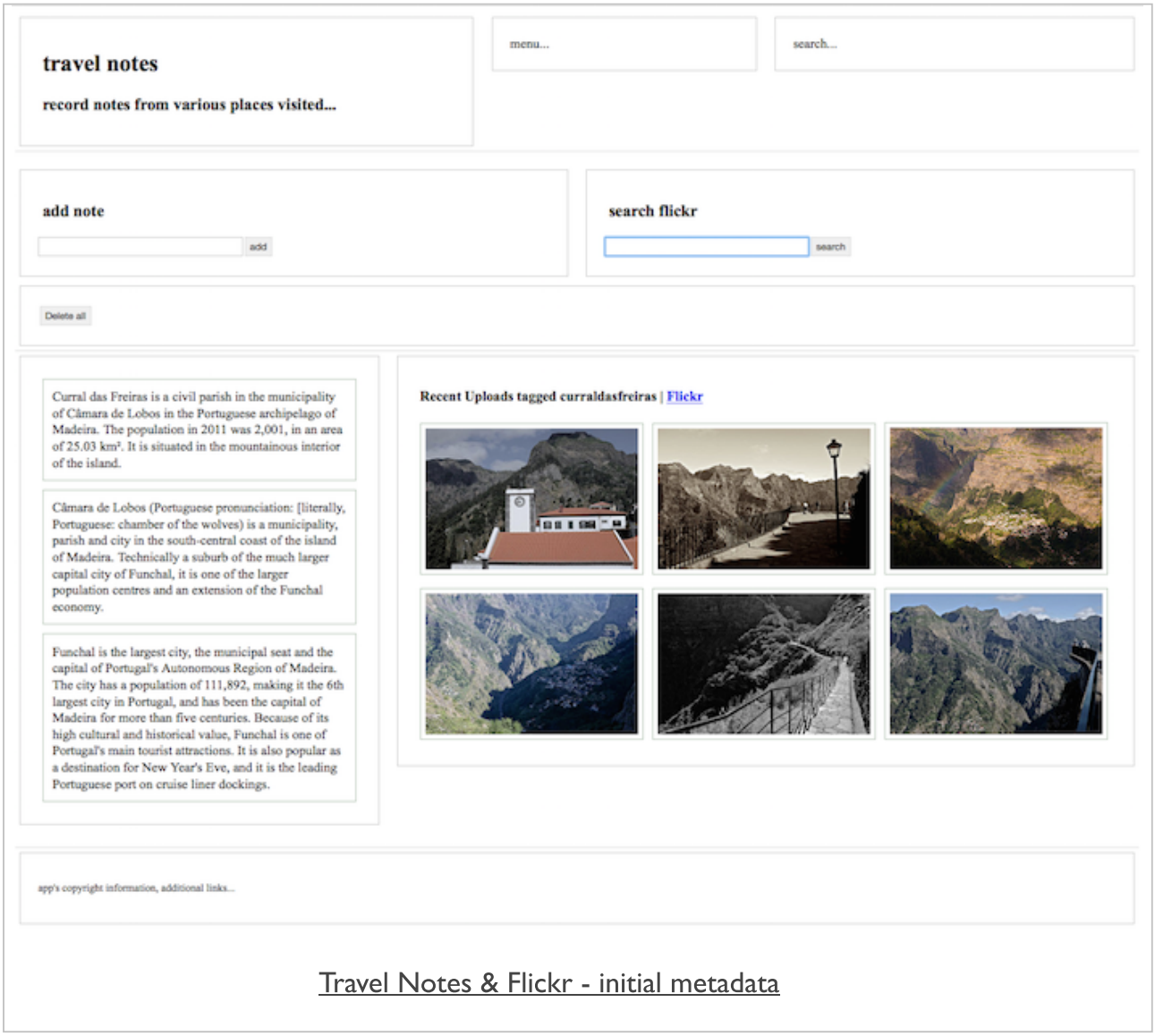
```
...
//create object for search metadata
var search_meta = {title:response.title, link:response.link};
...
```

- then pass this to a new function, called `metaOutput()`

```
//prepare and render metadata for returned search...
function metaOutput(data) {
  if (data !== "") {
  //search metadata from response
  var search_title = data.title;
  var search_link = data.link;
  //build heading output for metadata heading
  var metaHeading = '<h6>'+search_title+' | <a href="'+search_link+'">Flickr</a></h6>';
  //render metadata to contextual-output
  $(".contextual-output").prepend(metaHeading);
  }
}
```

- DEMO - travel notes & Flickr - initial metadata

Image - HTML5, CSS, & JS - Travel Notes & Flickr



Travel Notes & Flickr - initial metadata

HTML5, CSS, & JS - example - part 24

travel notes - basic refactoring of JS

- as we continue to add features and modify existing code
  - *may start to see unnecessary repetition and function calls in the code*
- eg: initial error handling for our contextual images
  - *`createImage()` function is being called in the `processImages()` function*
  - *called regardless of returned image data*
- `createImage()` is being used unnecessarily to manage the error handling
- move check to `processImages()` function
  - *then call function to render necessary error message*

```
function outputError(message) {
  var $img_error = $('<p class="flex-item error">').html(message);
  //check for visible contextual-output - if not visible
  if (checkVisible($(".contextual-output")) === true) {
    $(".note-output").removeClass("col-12");
    $(".note-output").addClass("col-4");
  }
  //append output to DOM
  $(".contextual-output").append($img_error);
  //fade in contextual-output with appended results
  $(".contextual-output").fadeIn("slow");
}
```

HTML5, CSS, & JS - example - part 25

travel notes - basic refactoring of JS

- updated `processImages()` function can call `.outputError()` function as needed

```
...
if (response.items.length !== 0) {
//logic to add metadata and each image...
}
else {
  var img_error = "No images available - please try a different search.";
  outputError(img_error);
}
...
```

- use this function to output error messages for any type of contextual data
- also remove some unnecessary replication of code
  - *by adding a simple function to change an element's class*

```
//modify element class - from, to
function changeClass(element, size1, size2) {
    $(element).removeClass(size1);
    $(element).addClass(size2);
}
```

- resize a class, for example to modify our grid output
  - *call this function - pass the selector to update, original class to remove, and new class to add*

HTML5, CSS, & JS - example - part 26

working with Flickr API - modify travel notes JS

- add a modification to check for the image loading and the notes
  - *offer status feedback to the user*

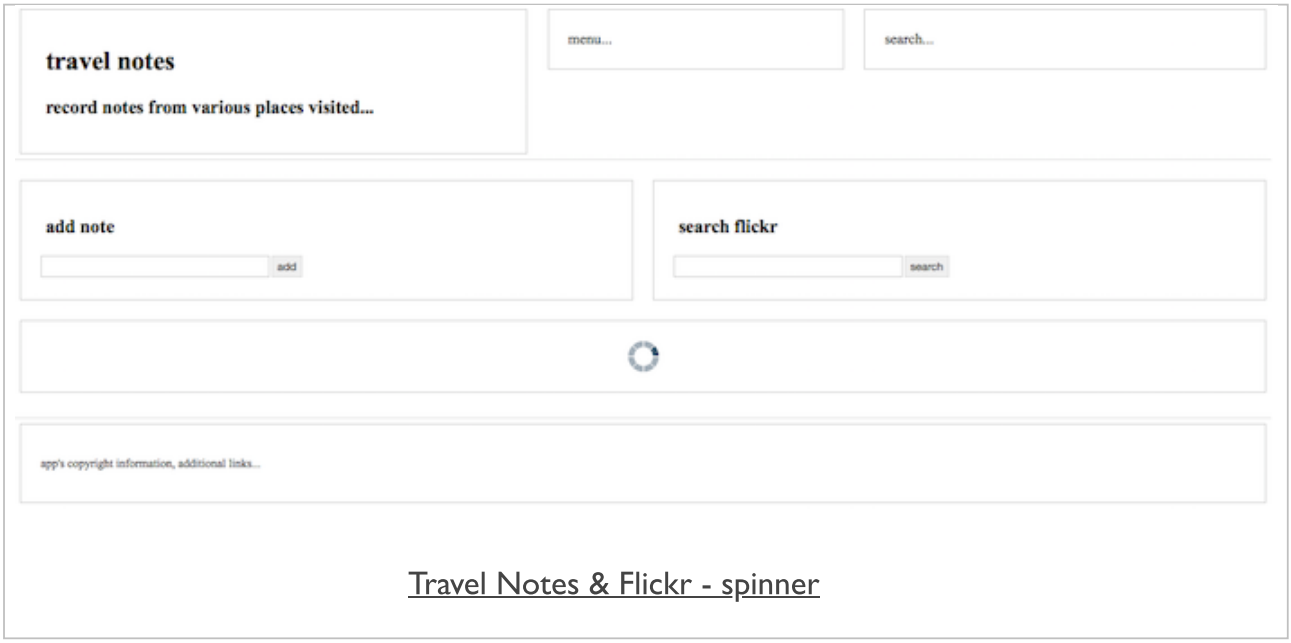
```
//add initial loader spinner for ajax...
$(".contextual-output").html('<img class="spinner" src="assets/images/ajax-loader.gif">');
```

- remove it when the deferred promise object has returned

```
//remove ajax spinner
$(".spinner").remove();
```

- DEMO - travel notes & Flickr - spinner

Image - HTML5, CSS, & JS - Travel Notes & Flickr



Travel Notes & Flickr - spinner

JS Server-side considerations - save data

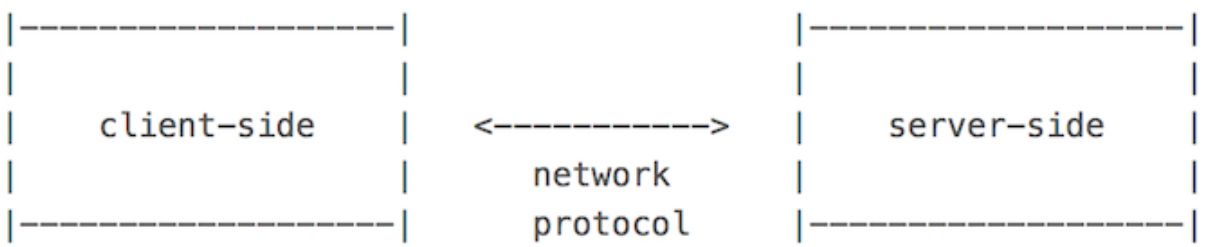
save JSON in travel notes app

- need to be able to save our simple notes
- now load from a JSON file as the app starts
  - *also we can add new notes, delete existing notes...*
- not as simple as writing to our existing JSON file direct from JS
  - *security implications if that was permitted directly from the browser*
- need to consider a few server-side options
- could use a combination of PHP on the server-side
  - *with AJAX jQuery on the client-side*
  - *traditional option with a simple ajax post to a PHP file on the server-side*
- consider JavaScript options on the client and server-side
- brief overview of working with **Node.js**

Server-side considerations - intro

- normally define computer programs as either client-side or server-side programs
- server-side programs normally abstract a resource over a network
  - *enabling many client-side programs to access at the same time*
  - *a common example is file requests and transfers*
- we can think of the client as the web browser
- a web server as the remote machine abstracting resources
- abstracts them via **hypertext transfer protocol**
  - *HTTP for short*
- designed to help with the transfer of HTML documents
  - *HTTP now used as an abstracted wrapper for many different types of resources*
  - *may include documents, media, databases...*

Image - Client-side and server-side computing



client-side & server-side

Server-side considerations - Node.js

intro - what is Node.js?

- Node.js is, in essence, a JavaScript runtime environment
  - *designed to be run outside of the browser*
- designed as a general purpose utility
- can be used for many different tasks including
  - *asset compilation*
  - *monitoring*
  - *scripting*
  - *web servers*
- with Node.js, role of JS is changing
  - *moving from client-side to a support role in back-end development*

Server-side considerations - Node.js

intro - speed of Node.js

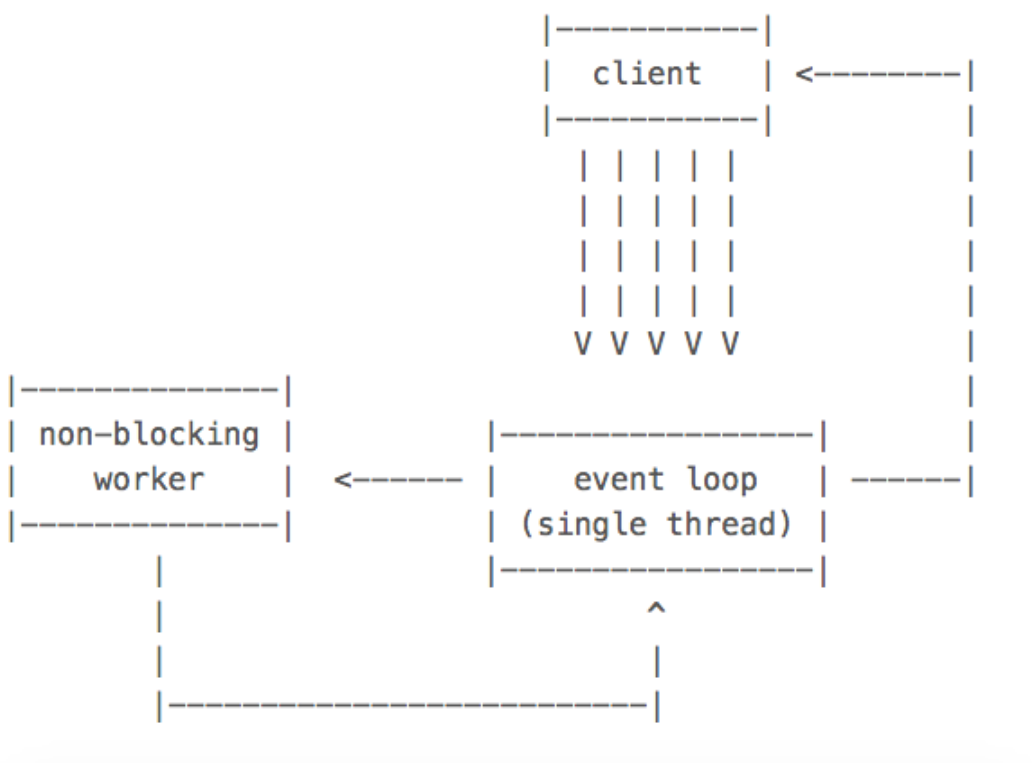
- a key advantage touted for Node.js is its speed
- many companies have noted the performance benefits of implementing Node.js
  - *including PayPal, Walmart, LinkedIn...*
- a primary reason for this speed boost is the underlying architecture of Node.js
- Node.js uses an **event-based** architecture
- instead of a threading model popular in compiled languages
- Node.js uses a single event thread by default
- all I/O is asynchronous

Server-side considerations - Node.js

intro - conceptual model for processing in Node.js

- how does Node.js, and its underlying processing model, actually work?
- client sends a hypertext transfer protocol, HTTP, request
  - *request or requests sent to Node.js server*
- event loop is then informed by the host OS
  - *passes applicable request and response objects as JavaScript closures*
  - *passed to associated worker functions with callbacks*
- long running jobs continue to run on various assigned worker threads
- responses are sent from the non-blocking workers back to the main event loop
  - *returned via a callback*
- event loop returns any results back to the client
  - *effectively when they're ready*

Image - Client-side and server-side computing



Node.js - conceptual model for processing

Server-side considerations - Node.js

intro - threaded architecture

- concurrency allows multiple things to happen at the same time
- common practice on servers due to the nature of multiple user queries
- Java, for example, will create a new thread on each connection
  - *threading is inherently resource expensive*
- size of a thread is normally around 4MB of memory
- naturally limits the number of threads that can run at the same time
- also inherently more complicated to develop platforms that are thread-safe
  - *thereby allowing for such functionality*
- due to this complexity
  - *many languages, eg: Ruby, Python, and PHP, do not have threads that allow for real concurrency*
  - *without custom binaries*
- JavaScript is similarly single-threaded
  - *able to run multiple code paths in parallel due to* ***events***

Server-side considerations - Node.js

intro - event-driven architecture

- JavaScript originally designed to work within the confines of the web browser
- had to handle restrictive nature of a single thread and single process for the whole page
- synchronous blocking in code would lock up a web page from all actions
  - *JavaScript was built with this in mind*
- due to this style of I/O handling
  - *Node.js is able to handle millions of concurrent requests on a single process*
- added, using libraries, to many other existing languages
  - *Akka for Java*
  - *EventMachine for Ruby*
  - *Twisted for Python*
  - ...
- JavaScript syntax already assumes events through its use of callbacks
- **NB:** if a query etc is CPU intensive instead of I/O intensive
  - *thread will be tied up*
  - *everything will be blocked as it waits for it to finish*

Server-side considerations - Node.js

intro - callbacks

- in most languages
  - *send an I/O query & wait until result is returned*
  - *wait before you can continue your code procedure*
- for example, submit a query to a database for a user ID
  - *server will pause that thread/process until database returns result for ID query*
- in JS, this concept is rarely implemented as standard
- in JS, more common to pass the I/O call a **callback**
- in JS, this **callback** will need to run when task is completed
  - *eg: find a user ID and then do something, such as output to a HTML element*
- biggest difference in these approaches
  - *whilst the database is fetching the user ID query*
  - *thread is free to do whatever else might be useful*
  - *eg: accept another web request, listen to a different event...*
- this is one of the reasons that Node.js returns good benchmarks and is easily scaled
- **NB:** makes Node.js well suited for I/O heavy and intensive scenarios

Server-side considerations - Node.js

install Node.js

- a number of different ways to install **Node.js**, **npm**, and the lightweight, customisable web framework **Express**
- run and test Node.js on a local Mac OS X or Windows machine
- download and install a package from the following URL
  - *Node.js - download*
- install the Node module, **Express**
- Express is a framework for web applications built upon Node.js
  - *minimal, flexible, & easily customised server*
- use *npm* to install the Express module

```
npm install -g express
```

- –g option sets a global flag for Express instead of limited local install
- installs Express command line tool
  - *allows us to start building our basic web application*
- now also necessary to install Express application generator

```
npm install -g express-generator
```

Server-side considerations - Node.js

NPM - intro

- **npm** is a package manager for Node.js
- Developers can use **npm** to share and reuse modules in Node.js applications
- **npm** can also be used to share complete Node.js applications
- example modules might include
  - *Markup, YAML etc parsers*
  - *database connectors*
  - *Express server*
  - ...
- **npm** is included with the default installers available at the Node.js website
- test whether **npm** is installed, simply issue the following command

```
npm
```

- should output some helpful information if **npm** is currently installed
- **NB:** on a Unix system, such as OS X or Linux
  - *best to avoid installing **npm** modules with `sudo` privileges*

Server-side considerations - Node.js

NPM - installing modules

- install existing **npm** modules, use the following type of command

```
npm install express
```

- this command installs module named `express` in the current directory
- it will act as a local installation within the current directory
- installing in a folder called `node_modules`
  - *this is the default behaviour for current installs*
- we can also specify a global install for modules
  - *eg: we may wish to install the **express** module with global scope*

```
npm install -g express
```

- again, the `-g` flag specifies the required global install

Server-side considerations - Node.js

NPM - importing modules

- import, or effectively add, modules in our Node.js code
  - *use the following declaration*

```
var module = require('express');
```

- when we run this application
  - *Node.js looks for the required module library and its source code*

Server-side considerations - Node.js

NPM - finding modules

- official online search tool for **npm** can be found at
  - *npmjs*
- top packages include options such as
  - *browserify*
  - *express*
  - *grunt*
  - *bower*
  - *karma*
  - ...
- also search for Node modules directly
  - *search from the command line using the following command*

```
npm search express
```

- returns results for module names and descriptions

Server-side considerations - Node.js

NPM - specifying dependencies

- ease Node.js app installation
  - *specify any required dependencies in an associated `package.json` file*
- allows us as developers to specify modules to install for our application
  - *which can then be run using the following command*

```
npm install
```

- helps reduce the need to install each module individually
- helps other users install an application as quickly as possible
- our application's dependencies are stored in one place
- example `package.json`

```
{
"name": "app",
"version": "0.0.1",
"dependencies": {
  "express": "4.2.x",
  "underscore": "-1.2.1"
}
}
```

Server-side considerations - Node.js

initial Express usage

- now use Express to start building our initial basic web application
- Express creates a basic shell for our web application
  - *cd to working directory and use the following command*

```
express /node/test-project
```

- command makes a new directory
  - *populates with required basic web application directories and files*
- cd to this directory and install any required dependencies,

```
npm install
```

- then run our new app,

```
npm start
```

- or run and monitor our app,

```
nodemon start
```

Server-side considerations - Node.js

initial Express server - setup

- we've now tested **npm**, and installed our first module with **Express**
- test **Express**, and build our first, simple server
- initial directory structure

```
|- .
   |- 424-node
      |- node_modules
```

- need to do is create a JS file to store our server code, so we'll add `server.js`

```
|- .
   |- 424-node
      |- node_modules
      |- server.js
```

- start adding our Node.js code to create a simple server

Server-side considerations - Node.js

initial Express server - server.js - part 1

- add some initial code to get our server up and running

```
/* a simple Express server for Node.js*/
var express = require("express"),
    http = require("http"),
    appTest;

// create our server - listen on port 3030
appTest = express();
http.createServer(appTest).listen(3030);

// set up routes
appTest.get("/test", function(req, res) {
  res.send("welcome to the 424 test app.");
});
```

- then start and test this server as follows at the command line

```
node server.js
```

Server-side considerations - Node.js

initial Express server - server.js - part 2

- open our web browser, and use the following URL

```
http://localhost:3030
```

- this is the route of our new server
  - *to get our newly created route use the following URL*

```
http://localhost:3030/test
```

- this will now return our specified route, and output message
- update our `server.js` file to support root directory level routes

```
appTest.get("/", function(req, res) {
  res.send("Welcome to the 424 server.")
});
```

- now load our server at the root URL

```
http://localhost:3030
```

- stop server from command line using `CTRL` and `c`

Server-side considerations - Node.js

initial Express server - server.js - part 3

- currently, initial Express server is managing some static routes for loading content
  - *we simply tell the server how to react when a given route is requested*
- what if we now want to serve some HTML pages?
  - *Express allows us to set up routes for static files*

```
//set up static file directory - default route for server
appTest.use(express.static(__dirname + "/app"));
```

- now defining Express as a static file server
  - *enabling us to publish our HTML, CSS, and JS files*
  - *published from our default directory, `/app`*
- if requested file not available
  - *server will check other available routes*
  - *or report error to browser if nothing found*
- DEMO - 424-node

Image - Client-side and server-side computing

welcome to the app file directory

simple Express server output

Server-side considerations - Node.js

working with data - JSON

- let us now work our way through a basic Node.js app
- serve our JSON, then read and load from a standard web app
- create our initial `server.js` file

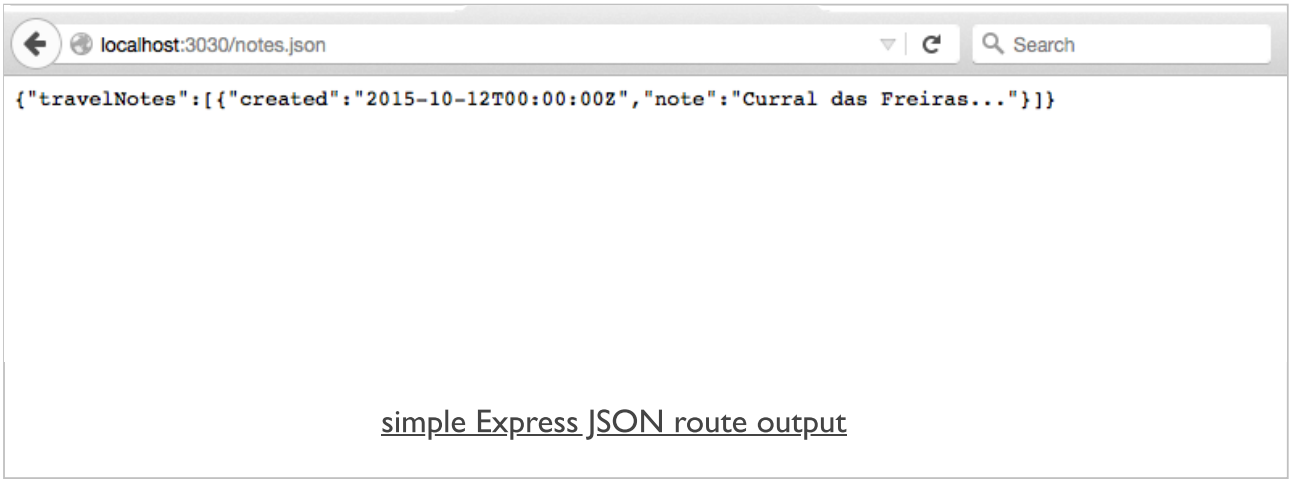
```
var express = require('express'),
    http = require("http"),
    jsonApp = express(),
    notes = {
      "travelNotes": [{
      "created": "2015-10-12T00:00:00Z",
      "note": "Curral das Freiras..."
      }]
    };

jsonApp.use(express.static(__dirname + "/app"));

http.createServer(jsonApp).listen(3030);

//json route
jsonApp.get("notes.json", function(req, res) {
  res.json(notes);
});
```

Image - Client-side and server-side computing



simple Express JSON route output

Server-side considerations - Node.js

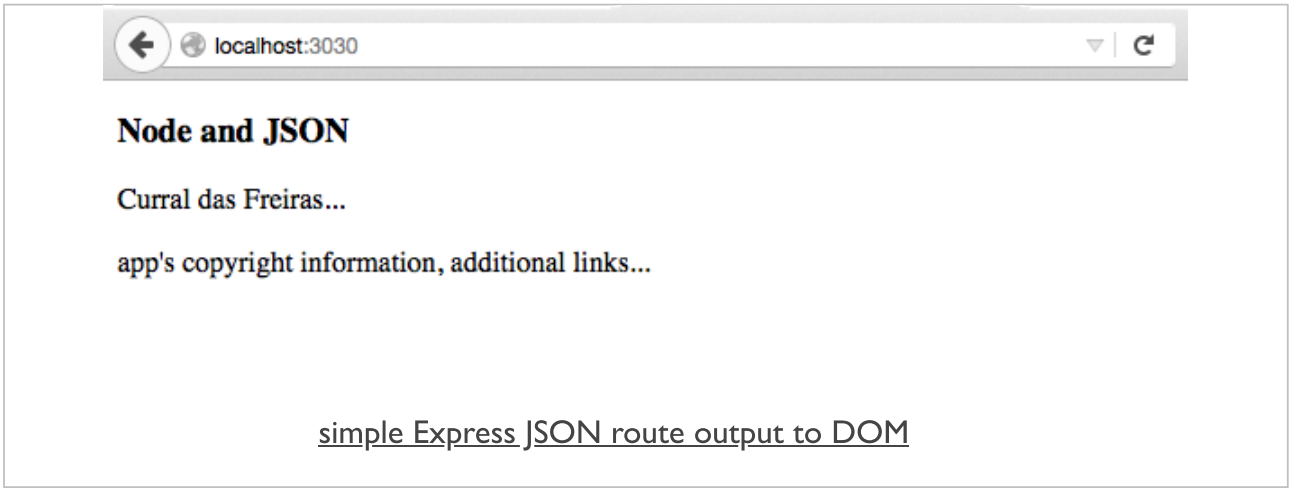
working with data - JSON

- now have our `get` routes setup for JSON
- now add some client-side logic to read that route
- render to the browser
- same basic patterns we've seen before
  - *using jQuery's `.getJSON()` function*

```
...
  $.getJSON("notes.json", function (response) {
    console.log("response = "+response.toSource());
    buildNote(response);
  })
...
```

- response object from our JSON
  - *this time from the server and not a file or API*
- use our familiar functions to create and render each note
  - *call our normal `buildNote()` function*
- DEMO - 424-node-json1

Image - Client-side and server-side computing



simple Express JSON route output to DOM

Demos

- Node.js
  - *424-node*
  - *424-node-json1*
  - *424-node-json2*
- 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*
  - *DEMO 2 - Travel Notes & Flickr*
  - *DEMO 3 - Travel Notes & Flickr - error checking*
  - *DEMO 4 - Travel Notes & Flickr - initial metadata*
  - *DEMO 5 - Travel Notes & Flickr - spinner*

Resources

- Chocolatey for Windows
- Chocolatey package manager for Windows
- Flickr
  - *Flickr API - Public feeds*
  - *Flickr API - Public feed - public photos & video*
- Homebrew for OS X
- Homebrew - the missing package manager for OS X
- MDN
  - *MDN - JS Objects*
- Node.js
- Node.js home
- Node.js - download
- ExpressJS
- W3
  - *W3 - CSS Flexible Box Layout Module 1*
  - *W3 - JS Object*
  - *W3 - JS Performance*