

# **Comp 324/424 - Client-side Web Design - Slides**

---

Spring Semester 2018

Dr Nick Hayward

## Final Assessment

---

- working final app
  - **NO** content management systems (CMSs) such as Drupal, Joomla, WordPress...
  - **NO** PHP, Python, Ruby, C# & .Net, Go, XML...
  - **NO** CSS frameworks, such as Bootstrap, Foundation, Materialize...
- presentation and demo - live working app...
  - due on Monday 23rd April 2018 @ 4.15pm
  - show and explain implemented differences from DEV week project
  - where and why did you update the app?
  - benefits of updates?
- how did you respond to peer review?
- final report
  - due on Monday 30th April 2018 @ 4.15pm

# Final report

---

- Final Report Outline

## 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 2I

---

### **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

record notes from various places visited...

menu...

search...

add note

search flickr

Delete all

Curral das Freiras is a civil parish in the municipality of Câmara de Lobos in the Portuguese archipelago of Madeira. The population in 2011 was 2,001, in an area of 25.03 km². It is situated in the mountainous interior of the island.

Câmara de Lobos (Portuguese pronunciation: [literally, Portuguese: chamber of the wolves]) is a municipality, parish and city in the south-central coast of the island of Madeira. Technically a suburb of the much larger capital city of Funchal, it is one of the larger population centres and an extension of the Funchal economy.

Funchal is the largest city, the municipal seat and the capital of Portugal's Autonomous Region of Madeira. The city has a population of 111,892, making it the 6th largest city in Portugal, and has been the capital of Madeira for more than five centuries. Because of its high cultural and historical value, Funchal is one of Portugal's main tourist attractions. It is also popular as a destination for New Year's Eve, and it is the leading Portuguese port on cruise liner dockings.

No images available. Please try a different search.

app's copyright information, additional links...

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

record notes from various places visited...

menu...

search...

add note

search flickr







Delete all

Curral das Freiras is a civil parish in the municipality of Câmara de Lobos in the Portuguese archipelago of Madeira. The population in 2011 was 2,001, in an area of 25.03 km². It is situated in the mountainous interior of the island.

Câmara de Lobos (Portuguese pronunciation: [literally, Portuguese: chamber of the wolves] is a municipality, parish and city in the south-central coast of the island of Madeira. Technically a suburb of the much larger capital city of Funchal, it is one of the larger population centres and an extension of the Funchal economy.

Funchal is the largest city, the municipal seat and the capital of Portugal's Autonomous Region of Madeira. The city has a population of 111,892, making it the 6th largest city in Portugal, and has been the capital of Madeira for more than five centuries. Because of its high cultural and historical value, Funchal is one of Portugal's main tourist attractions. It is also popular as a destination for New Year's Eve, and it is the leading Portuguese port on cruise liner dockings.

Recent Uploads tagged curraldasfreiras | Flickr



app's copyright information, additional links...

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('');
```

- 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

record notes from various places visited...

meta...

search...

add note

search flickr

app's copyright information, additional links...

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

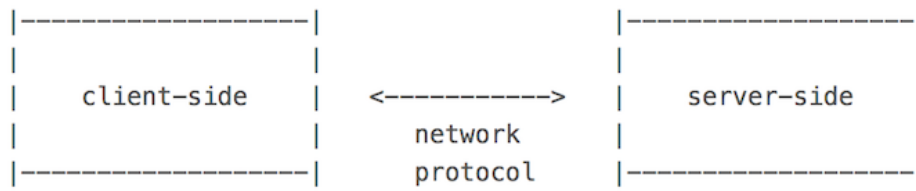
---

- 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

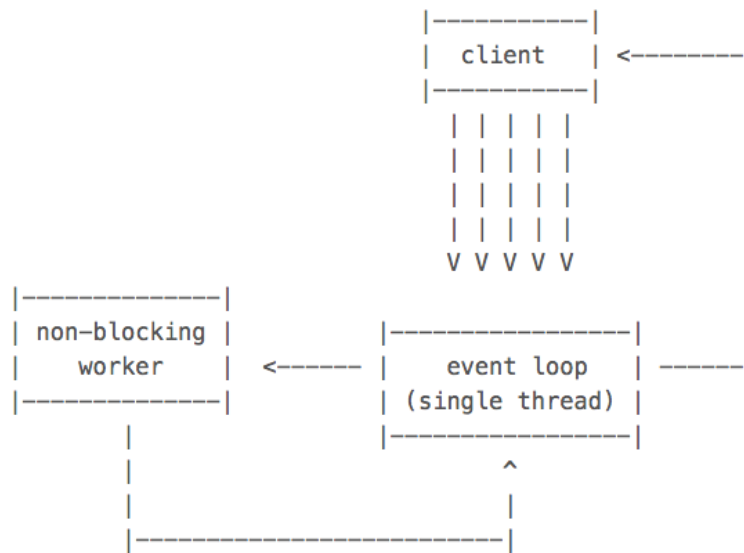
# Server-side considerations - Node.js

---

## **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 I*

- 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

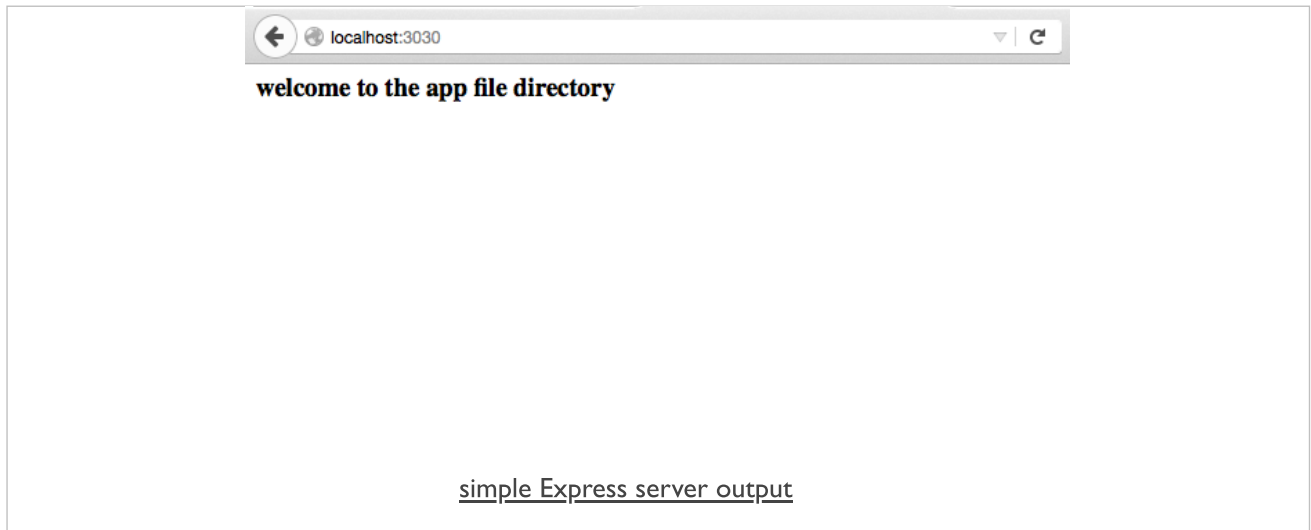
- currently, initial Express server is managing some static routes for loading content
  - we simply tell the server how to react when a given route is requested
- what if we now want to serve some HTML pages?
  - Express allows us to set up routes for static files

```
//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

---



## 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..."
      }]
    };

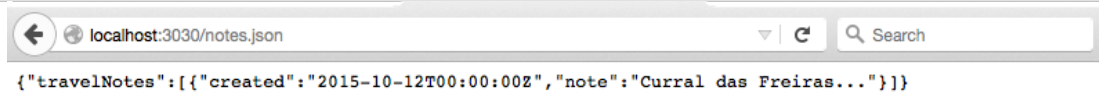
jsonApp.use(express.static(__dirname + "/app"));

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

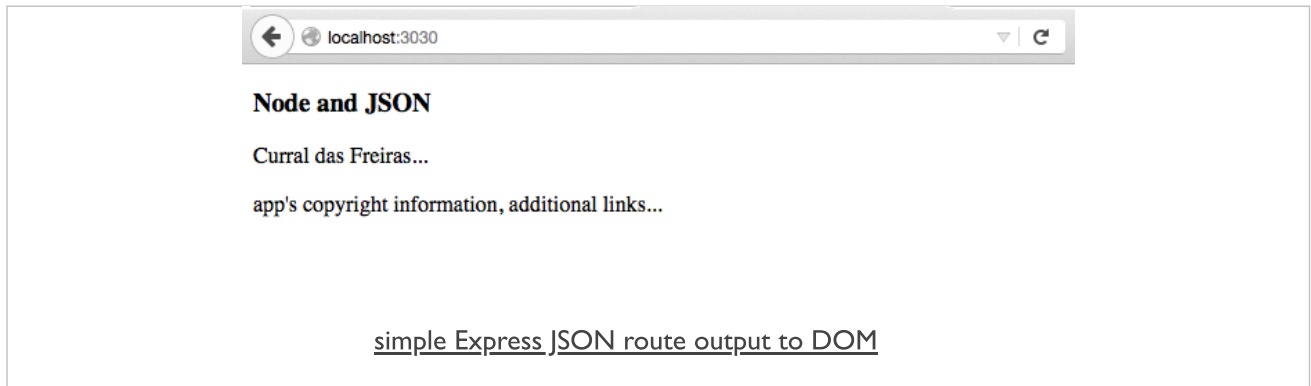
- now have our get routes setup for JSON
- now add some client-side logic to read that route
- render to the browser
- same basic patterns we've seen before
  - using jQuery's `.getJSON( )` function

```
...  
$.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-json |

## Image - Client-side and server-side computing

---





## Server-side considerations - Node.js

---

### working with data - post data

- we've seen examples that load JSON data
  - using jQuery's `.getJSON( )` function
- now consider jQuery's `post` function
  - allow us to easily send JSON data to the server
  - simply called *post*
- begin our updates by creating a new route in our Express server
  - one that will handle the *post* route

```
jsonApp.post("/notes", function(req, res) {  
  //return simple JSON object  
  res.json({  
    "message": "post complete to server"  
  });  
});
```

## Server-side considerations - Node.js

---

### ***working with data - post data***

- may look similar to our earlier `get` routes
  - *difference due to browser restrictions*
  - *can't simply request direct route using our browser*
  - *as we did with `get` routes*
- need to change JS we use for the client-side
  - *allows us to post new route*
  - *then enables view of the returned message*
- update our test app to store data on the server
  - *then initialise our client with this stored data*

## Server-side considerations - Node.js

---

### working with data - post data

- start with a simple check that the post route is working correctly
  - *add a button, submit a request to the post route, and then wait for the response*
  - *add event handler for a button*

```
$("#post").on("click", function() {  
  $.post("notes", {}, function (response) {  
    console.log("server post response returned..." + response.toSource());  
  })  
});
```

- submit a post request
  - *specify the route for the post to the Node.js server*
  - *then specify the data to post - an empty object in this example*
  - *the specify a callback for the server's response*
- test returns the following output to the browser's console,

```
server post response returned...({message:"post complete to server"})
```

## Server-side considerations - Node.js

---

### *working with data - post data*

- now send some data to the server
  - *add new note to our object*
- update the server to handle this incoming object
  - *process the submitted jQuery JSON into a JavaScript object*
  - *ready for use with the server*
- use the **Express** module's `body-parser` plugin
- update `server.js` as follows

```
//add body-parser for JSON parsing etc...
var bodyParser = require("body-parser");
...
//Express will parse incoming JSON objects
jsonApp.use(bodyParser.urlencoded({ extended: false }));
...
```

- as server receives new JSON object
  - *it will now parse, or process, this object*
  - *ensures it can be stored on the server for future use*

## Server-side considerations - Node.js

---

### working with data - post data

- now update our test button's event handler
  - send a new note as a JSON object
- note will retrieve its new content from the input field
  - gets the current time from the node server

```
$(".note-input button").on("click", function() {  
  //get values for new note  
  var note_text = $(".note-input input").val();  
  var created = new Date();  
  //create new note  
  var newNote = {"created":created, "note":note_text};  
  //post new note to server  
  $.post("notes", newNote, function (response) {  
    console.log("server post response returned..." + response.toSource());  
  })  
});
```

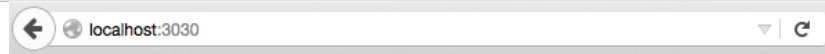
- input field and button follow the same pattern as previous examples

```
<!-- note input -->  
<section class="note-input col-6">  
  <h5>add note</h5>  
  <input><button>add</button>  
</section>
```

- DEMO - 424-node-json2

# Image - Client-side and server-side computing

---



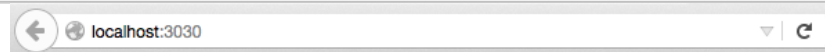
## Node and JSON

add note

Curral das Freiras...

app's copyright information, additional links...

Node.js and Express - post new note to server



## Node and JSON

add note

Curral das Freiras...

new note for the server

app's copyright information, additional links...

Node.js and Express - get new notes from server

## Server-side considerations - data storage

---

### *intro*

- tested Node.js, created a server for hosting our files and routes with ExpressJS
  - *read JSON from the server*
  - *updated our JSON on the server-side*
- works well as long as we do not need to restart, repair, update etc our server
- data lost with restart etc...
- need to consider a persistent data storage
  - *independent from the application*
- NoSQL options such as Redis and MongoDB
- integration with Node.js

## Server-side considerations - data storage

---

### SQL or NoSQL

- common database usage and storage
  - *often thought solely in terms of SQL, or structured query language*
- SQL used to query data in a relational format
- relational databases, for example MySQL or PostgreSQL, store their data in tables
  - *provides a semblance of structure through rows and cells*
  - *easily cross-reference, or relate, rows across tables*
- a relational structure to map authors to books, players to teams...
  - *thereby dramatically reducing redundancy, required storage space...*
- improvement in storage capacities, access...
  - *led to shift in thinking, and database design in general*
- started to see introduction of non-relational databases
  - *often referred to simply as **NoSQL***
- with NoSQL DBs
  - *redundant data may be stored*
  - *such designs often provide increased ease of use for developers*
- some NoSQL examples for specific use cases
  - *eg: fast reading of data more efficient than writing*
  - *specialised DB designs*



## Server-side considerations - data storage

---

### Redis - intro

- Redis provides an excellent example of NoSQL based data storage
- designed for fast access to frequently requested data
- improvement in performance often due to a reduction in perceived reliability
  - *due to in-memory storage instead of writing to a disk*
- able to flush data to disk
  - *performs this task at given points during uptime*
  - *for majority of cases considered an in-memory data store*
- stores this data in a **key-value** format
  - *similar in nature to standard object properties in JavaScript*
- Redis often a natural extension of conventional data structures
- Redis is a good option for quick access to data
  - *optionally caching temporary data for frequent access*

## Server-side considerations - data storage

---

### Redis - installation

- On OS X, use the Homebrew package manager to install Redis

```
brew install redis
```

- Windows port maintained by the Microsoft Open Tech Group - Redis
  - or use Windows package manager - <https://chocolatey.org/>
- for Linux - download, extract, and compile Redis

```
$ wget http://download.redis.io/releases/redis-3.0.5.tar.gz
$ tar xzf redis-3.0.5.tar.gz
$ cd redis-3.0.5
$ make
```

## Server-side considerations - data storage

---

### Redis - server and CLI

- start the Redis server with the following command,

```
redis-server
```

- interact with our new server directly using the CLI tool,

```
redis-cli
```

- store some data in Redis using the set command
  - *create a new key for `notes`, and then set its value to 0*
  - *if value is set, Redis returns `OK`*

```
set notes 0
```

- retrieve a value using the get command
  - *returns our set value of 0*

```
get notes
```

## Image - Client-side and server-side computing

---

```
Drs-MacBook-Air-2:~ ancientlives$ redis-cli  
127.0.0.1:6379> set notes 0  
OK  
127.0.0.1:6379> get notes  
"0"  
127.0.0.1:6379> █
```

Redis CLI - set and get

## Server-side considerations - data storage

---

### Redis - server and CLI

- also manipulate existing values for a given key
  - eg: *increment and decrement a value, or simply delete a key*
- increment key notes value by 1

```
incr notes
```

- decrement key notes value by 1

```
decr notes
```

- we can then increment or decrement by a specified amount

```
// increment by 10
incrby notes 10
// decrement by 5
decrby notes 5
```

- delete our key

```
// single key deletion
del notes
// multiple keys deletion
del notes notes2 notes3
```

## Image - Client-side and server-side computing

---

```
Drs-MacBook-Air-2:~ ancientlives$ redis-cli
127.0.0.1:6379> set notes 0
OK
127.0.0.1:6379> get notes
"0"
127.0.0.1:6379> incr notes
(integer) 1
127.0.0.1:6379> incr notes
(integer) 2
127.0.0.1:6379> get notes
"2"
127.0.0.1:6379> decr notes
(integer) 1
127.0.0.1:6379> get notes
"1"
127.0.0.1:6379> incrby notes 10
(integer) 11
127.0.0.1:6379> get notes
"11"
127.0.0.1:6379> decrby notes 5
(integer) 6
127.0.0.1:6379> get notes
"6"
```

Redis CLI - increment and decrement

## Server-side considerations - data storage

---

### ***Redis and Node.js setup***

- test Redis with our Node.js app
- new test app called 424-node-redis1

```
| - 424-node-redis1  
  | - app  
    | - assets  
  | - node_modules  
  | - package.json  
  | - server.js
```

- create new file, `package.json` to track project
  - eg: *dependencies, name, description, version...*

## Server-side considerations - data storage

---

### *Redis and Node.js - package.json*

```
{
  "name": "424-node-redis1",
  "version": "1.0.0",
  "description": "test app for node and redis",
  "main": "server.js",
  "dependencies": {
    "body-parser": "^1.14.1",
    "express": "^4.13.3",
    "redis": "^2.3.0"
  },
  "author": "ancientlives",
  "license": "ISC"
}
```

- we can write the `package.json` file ourselves or use the interactive option

```
npm init
```

- then add extra dependencies, eg: Redis, using

```
npm install redis --save
```

- use `package.json` to help with app management and abstraction...



## Server-side considerations - data storage

---

### *Redis and Node.js - set notes value*

- add Redis to our earlier test app
- import and use Redis in the `server.js` file

```
...  
var express = require("express"),  
    http = require("http"),  
    bodyParser = require("body-parser"),  
    jsonApp = express(),  
    redis = require("redis");  
...
```

- create client to connect to Redis from Node.js

```
//create client to connect to Redis  
redisConnect = redis.createClient();
```

- then use Redis, for example, to store access total for notes on server

```
redisConnect.incr("notes");
```

- check Redis command line for change in notes value

```
get notes
```

## Server-side considerations - data storage

---

### Redis and Node.js - get notes value

- now set the counter value for our notes
  - *add our counter to the application to record access count for notes*
- use the get command with Redis to retrieve the incremented values for the notes key

```
redisConnect.get("notes", function(error, notesCounter) {  
  //set counter to int of value in Redis or start at 0  
  notesTotal.notes = parseInt(notesCounter,10) || 0;  
});
```

- get accepts two parameters - error and return value
- Redis stores values and strings
  - *convert string to integer using `parseInt()`*
  - *two parameters - return value and `base-10` value of the specified number*
- value is now being stored in a global variable `notesTotal`
  - *declared in `server.js`*

```
var express = require("express"),  
    http = require("http"),  
    bodyParser = require("body-parser"),  
    jsonApp = express(),  
    redis = require("redis"),  
    notesTotal = {};
```

## Server-side considerations - data storage

---

### Redis and Node.js - get notes value

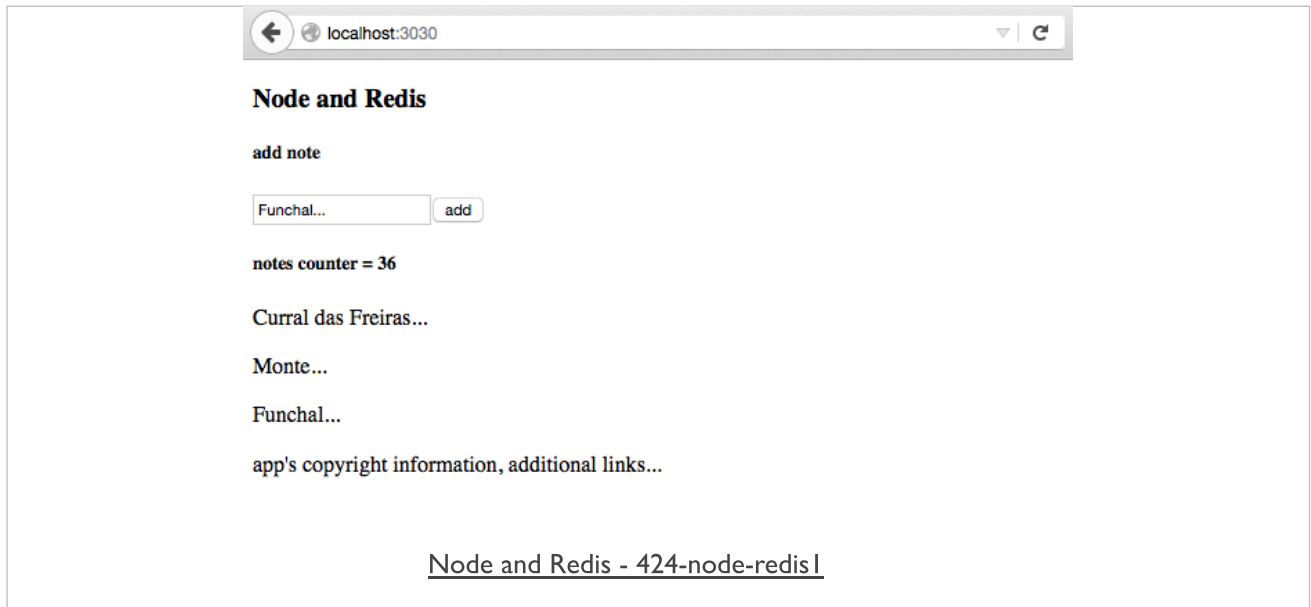
- store notes counter value in Redis
- create new route in `server.js`
  - *monitor the returned JSON for the counter*

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

- start using it with our application
  - *load by default, within event handler...*
- render to DOM
- store as a internal log record
- link to create note event handler...
- DEMO - 424-node-redis I

# Image - Client-side and server-side computing

---



## Server-side considerations - data storage

---

### **MongoDB - intro**

- MongoDB is another example of a NoSQL based data store
  - *a database that enables us to store our data on disk*
- unlike MySQL, for example, it is not in a relational format
- MongoDB is best characterised as a **document-oriented** database
- conceptually may be considered as storing objects in collections
- stores its data using the BSON format
  - *consider similar to JSON*
  - *use JavaScript for working with MongoDB*

## Server-side considerations - data storage

---

### **MongoDB - document oriented**

- SQL database, data is stored in tables and rows
- MongoDB, by contrast, uses **collections** and **documents**
- comparison often made between a collection and a table
- **NB:** a document is quite different from a table
- a document can contain a lot more data than a table
- a noted concern with this document approach is duplication of data
- one of the trade-offs between NoSQL (MongoDB) and SQL
- SQL - goal of data structuring is to normalise as much as possible
- thereby avoiding duplicated information
- NoSQL (MongoDB) - provision a data store, as easy as possible for the application to use

## Server-side considerations - data storage

---

### **MongoDB - BSON**

- BSON is the format used by MongoDB to store its data
- effectively, JSON stored as binary with a few notable differences
  - eg: *ObjectId* values - data type used in MongoDB to uniquely identify documents
  - created automatically on each document in the database
  - often considered as analogous to a primary key in a SQL database
- *ObjectId* is a large pseudo-random number
- for nearly all practical occurrences, assume number will be unique
- might cease to be unique if server can't keep pace with number generation...
- other interesting aspect of *ObjectId*
  - they are *partially* based on a timestamp
  - helps us determine when they were created

# Server-side considerations - data storage

---

## **MongoDB - general hierarchy of data**

- in general, MongoDB has a three tiered data hierarchy

1. database

- *normally one database per app*
- *possible to have multiple per server*
- *same basic role as DB in SQL*

2. collection

- *a grouping of similar pieces of data*
- *documents in a collection*
- *name is usually a noun*
- *resembles in concept a table in SQL*
- *documents do not require the same schema*

3. document

- *a single item in the database*
- *data structure of field and value pairs*
- *similar to objects in JSON*
- *eg: an individual user record*



# Demos

---

## Travel notes app - series 4

- DEMO 4 - Travel Notes & Flickr - initial metadata
- DEMO 5 - Travel Notes & Flickr - spinner

## Node.js

- 424-node
- 424-node-json1
- 424-node-json2

## Redis

- 424-node-redis1

## References

---

- Chocolatey for Windows
  - *Chocolatey package manager for Windows*
- Homebrew for OS X
  - *Homebrew - the missing package manager for OS X*
- MongoDB
  - *MongoDB - For Giant Ideas*
- Node.js
  - *Node.js home*
  - *Node.js - download*
  - *ExpressJS*
  - *ExpressJS body-parser*
- Redis
  - *redis.io*
  - *redis commands*
  - *redis - npm*
  - *try redis*
  - *Windows support*
- Various
  - *Create your own AJAX loader*