Comp 388/424 - Client-side Web Design

Fall Semester 2015 - Week 5

Dr Nick Hayward

Contents

JS core - `this`

- `this` keyword - correct and appropriate usage
  - *commonly misunderstood feature of JS*
- value of `this` is not inherently linked with the function itself
- value of `this` determined in response to how the function is called
- value itself can be dynamic, simply based upon how the function is called
- if a function contains `this`, its reference will usually point to an **object**
- manipulate and update the underlying context using `.apply()`, `.bind()`, and `.call()`

JS core - `this` default - part I

global, window object

- when we call a function, we can bind the `this` value to the `window` object
- resultant object refers to the root, in essence the `global` scope

```
function test1() {
  console.log(this);
}

test1();
```

- **NB:** the above will return a value of `undefined` in strict mode.
- also check for the value of `this` relative to the global object,

```
var a = 49;

function test1() {
    console.log(this.a);
}

test1();
```

- JSFiddle - this - window
- JSFiddle - this - global

JS core - `this` default - part 2

object literals

- within an object literal, the value of `this`, thankfully, will always refer to its own object

```
var object1 = {
    method: test1
};

function test1() {
    console.log(this);
}

object1.method();
```

- return value for `this` will be the object itself
- we get the returned object with a property and value for the defined method
- other object properties and values will be returned and available as well
- JSFiddle - this - literal
- JSFiddle - this - literal 2

JS core - `this` default - part 3

events

- for events, value of `this` points to the owner of the bound event

```
<div id="test">click to test...</div>
```

```
var testDiv = document.getElementById('test');

function output() {
  console.log(this);
};

testDiv.addEventListener('click', output, false);
```

- element is clicked, value of `this` becomes the clicked element
- also change the context of `this` using built-in JS functions
  - *such as `.apply()`, `.bind()`, and `.call()`*
- JSFiddle - this - events

JS extras - best practices - part 1

a few best practices...

variables

- limit use of global variables in JavaScript
  - *easy to override*
  - *can lead to unexpected errors and issues*
  - *should be replaced with appropriate local variables, closures*
- local variables should always be declared with keyword `var`
  - *avoids automatic global variable issue*

declarations

- add all required declarations at the top of the appropriate script or file
  - *provides cleaner, more legible code*
  - *helps to avoid unnecessary global variables*
  - *avoid unwanted re-declarations*

types and objects

- avoid declaring numbers, strings, or booleans as objects
- treat more correctly as primitive values
  - *helps increase the performance of our code*
  - *decrease the possibility for issues and bugs*

JS extras - best practices - part 2

type conversions and coercion

- weakly typed nature of JS
  - *important to avoid accidentally converting one type to another*
  - *converting a number to a string or mixing types to create a NaN (Not a Number)*
- often get a returned value set to NaN instead of generating an error
  - *try to subtract one string from another may result in* `NaN`

comparison

- better to try and work with === instead of ==
  - *== trys to coerce a matching type before comparison*
  - *=== forces comparison of values and type*

defaults

- when parameters are required by a function
  - *function call with a missing argument can lead to it being set as* ***undefined***
  - *good coding practice to assign default values to arguments*
  - *helps prevent issues and bugs*

switches

- consider a `default` for the switch conditional statement
- ensure you always set a `default` to end a switch statement

JS extras - performance - part 1

loops

- try to limit the number of calculations, executions, statements performed per loop iteration
- check loop statements for assignments and statements
  - *those checked or executed once*
  - *rather than each time a loop iterates*
- `for` loop is a standard example of this type of quick optimisation

```
// bad
for (i = 0; i < arr.length; i++) {
...
}
// good
l = arr.length;
for (i = 0; i < l; i++) {
...
}
```

- source - W3

JS extras - performance - part 2

DOM access

- repetitive DOM access can be slow, and resource intensive
- try to limit the number of times code needs to access the DOM
- simply access once and then use as a local variable

```
var testDiv = document.getElementById('test');
testDiv.innerHTML = "test...";
```

JavaScript loading

- not always necessary to place JS files in the `<head>` element
- adding JS scripts to end of the page's body
  - *allows browser to load the page first*
- HTTP specification defines browsers should not download more than two components in parallel

JS extras - JSON - part 1

- JSON is a lightweight format and wrapper for storing and transporting data
- inherently language agnostic, easy to read and understand
- growing rapidly in popularity
  - *many online APIs have updated XML to JSON for data exchange*
- syntax of JSON is itself derived from JS object notation
  - *text-only format*
- allows us to easily write, describe, and manipulate JSON in practically any programming language
- **JSON syntax** follows a few basic rules,
  - *data is recorded as name/value pairs*
  - *data is separated by commas*
  - *objects are defined by a start and end curly brace*
  - *{ }*
  - *arrays are defined by a start and end square bracket*
  - *[ ]*

JS extras - JSON - part 2

- underlying construct for JSON is a pairing of name and value

```
"city":"Marseille"
```

JSON Objects

- contained within curly braces
- objects can contain multiple name/value pairs

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

JS extras - JSON - part 3

JSON Arrays

- contained within square brackets
  - *arrays can also contain objects.*

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

- use this with JavaScript, and parse the JSON object.
  - *JSFiddle - Parse JSON*

HTML5, CSS, & JS - example - part 1

Structure

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

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

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

HTML5, CSS, & JS - example - part 2

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>travel notes - v0.1</title>
    <meta name="description" content="information on travel destinations">
    <meta name="author" content="ancientlives">
    <!-- css styles... -->
    <link rel="stylesheet" type="text/css" href="assets/styles/style.css">
  </head>
  <body>
    ...
    <!-- js scripts... -->
    <script type="text/javascript"      src="https://code.jquery.com/jquery-2.1.4.min.js">
    <script type="text/javascript" src="assets/scripts/travel.js"></script>
  </body>
</html>
```

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

HTML5, CSS, & JS - example - part 3

index.html - body

```
<body>
  <!-- document header -->
  <header>
    <h3>travel notes</h3>
    <p>record notes from various cities and placed visited...</p>
  </header>
  <!-- document main -->
  <main>
    <!-- note input -->
    <section class="note-input">
    </section>
    <!-- note output -->
    <section class="note-output">
    </section>
  </main>
  <!-- document footer -->
  <footer>
    <p>app's copyright information, additional links...</p>
  </footer>
  <!-- js scripts... -->
  <script type="text/javascript" src="https://code.jquery.com/jquery-2.1.4.min.js"></scrip
  <script type="text/javascript" src="assets/scripts/travel.js"></script>
</body>
```

HTML5, CSS, & JS - example - part 4

style.css

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

HTML5, CSS, & JS - example - part 5

travel.js

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

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

};

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

- a simple JS function to hold the basic logic for our app
- call this function any reasonable, logical name
- in initial function, we set the `strict` pragma
- add an example call to the jQuery function, `html()`
  - *sets some initial note content*
- function `travelNotes()` loaded using the jQuery function `ready()`
  - *many different ways to achieve this basic loading of app logic*
- DEMO 1 - travel notes - v0.1

HTML5, CSS, & JS - example - part 6

add a note

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

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

HTML5, CSS, & JS - example - part 7

tidy up styling

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

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

HTML5, CSS, & JS - example - part 8

input update

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

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

- also update css for `input` and `button`
- remove button's rounded borders to match style of `input`
- match border for button to basic design aesthetics
- set cursor appropriate for a link style...
- DEMO 2 - travel notes - v0.2

HTML5, CSS, & JS - example - part 9

interaction - add a note

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

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

HTML5, CSS, & JS - example - part 10

interaction - add a note - output

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

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

- output some static text to `note-output`
- DEMO 3 - travel notes - v0.3

HTML5, CSS, & JS - example - part 11

interaction - add a note - output

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

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

};

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

- DEMO 4 - travel notes - v0.4

HTML5, CSS, & JS - example - part 12

interaction - add a note - clear input

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

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

};

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

- DEMO 5 - travel notes - v0.5

HTML5, CSS, & JS - example - part 13

interaction - add a note - keyboard listener

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

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

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

HTML5, CSS, & JS - example - part 14

interaction - add a note - abstract code

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

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

HTML5, CSS, & JS - example - part 15

interaction - add a note - travel.js

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

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

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

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

- DEMO 6 - travel notes - v0.6

HTML5, CSS, & JS - example - part 16

interaction - add a note - animate

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

HTML5, CSS, & JS - example - part 17

interaction - add a note - animate js

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

- DEMO 7 - travel notes - v0.7

HTML5, CSS, & JS - example - part 18

style and render notes

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

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

- DEMO 8 - travel notes - v0.8

HTML5, CSS, & JS - final thoughts

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

jQuery - basics - part 1

intro

- jQuery offers us a number of useful tools and options for building web apps
- packaged, prepared JavaScript library
  - *a lot easier to work with, and develop for, than standard JavaScript*
- features simpler syntax and a concise set of options for manipulating the DOM
  - *often simply quicker and easier to write our apps with jQuery than JavaScript*
- jQuery is an inherently expressive approach to working with JavaScript
  - *in particular, manipulating the DOM*
- consistent approach to handling events in the DOM
- includes useful, simplified approach to adding AJAX functionality

jQuery - basics - part 2

selectors

- jQuery works with selectors using a similar concept as CSS
- we can use CSS selectors as a jQuery selector

```
$("div")
$("p")
$(".note-input")
$(".note-input button")
$("p:nth-child(even)")
...
```

- jQuery may share many selectors with CSS
  - *some cases where jQuery will slightly differ*
- adds useful set of pseudoclasses and pseudoelements not in CSS

```
$("p:parent")
```

- use the above to find all paragraphs with children, including text
- a jQuery extension, and not part of the CSS specification

jQuery - basics - part 3

manipulate the DOM

```
<body>
  <!-- document header -->
  <header>
    <h3></h3>
    <p></p>
  </header>
  <!-- document main -->
  <main>
    <!-- note input -->
    <section class="note-input">
      <h5>add note</h5>
      <input><button></button>
    </section>
    <!-- note output -->
    <section class="note-output">
    </section>
  </main>
  <!-- document footer -->
  <footer>
    <p></p>
  </footer>
</body>
```

- benefits of using jQuery is the ease it offers for manipulating the DOM
- add elements, delete them, move them around...

jQuery - basics - part 4

add elements

- add a new element to our app
  - *simply append or prepend to a given positon in the DOM*

```
//append note text to note-output
$(".note-output").append($note);
```

- adds our new element, and content to the DOM
  - *end of the selected element in document*

```
//append note text to note-output
$(".note-output").prepend($note);
```

- prepend to the document
  - *adds to the end of the selected element*
- additional options in JQuery, such as `prependTo()`
- differ slightly on the target for the content
- useful to select an element, then add to another elsewhere in DOM

jQuery - basics - part 5

remove elements

- also remove elements from the DOM
- easiest option is to use the `remove()` function on a given selector

```
$("p:nth-child(even)").remove();
```

- also empty an element, remove all child elements from selected element
  - *remove all of the notes, those we added in paragraph elements*

```
$(".note-output p").empty();
```

- also temporarily remove elements from the window

```
$note.fadeOut("slow");
```

- elements are not removed from the DOM, their style is updated

```
display: none;
```

jQuery - basics - part 6

events and async

- jQuery uses a standard pattern for events and handling

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

- allows us to set up listeners for many user triggered events
- commonly known as **event-driven** or **asynchronous** programming
- main difference with more traditional procedural patterns, is the way we use **callbacks**
  - *allow us to set functions for later execution*
- functions are set as parameters, then executed at the appropriate, required time
- callbacks are not only appropriate for interaction or user events
- use them throughout our programming to schedule functions and execution

```
setTimeout(function() {
  ...
}, 2000);
```

- an issue with **asynchronous** programming
  - *often simply being aware of the execution order or sequence of events*

Design and interface - intro

- consider some of the concepts, challenges, and options for interface design on the client-side
- important to remember the very nature of these applications
- these apps are inherently
  - *highly interactive*
  - *display content from myriad sources, including databases and streaming APIs*
  - *communicate with other systems*
  - *now more dynamic than ever*
- often designed and implemented with more than one activity in mind
- often represent actions such as finding results and records
  - *whilst also managing that data*
- access to and awarenesss of real-time data and streams
  - *strongly influencing our design and development*
  - *from news to banking*

Design and interface - goals

- an issue with app design is often focusing on both functionality, and complementary aesthetics
- look at client-side design trends in general
  - *ubiquity of digital applications has led to a reduction for many early design conventions*
  - *rare to see a site still use browser defaults for links*
  - *actual design or aesthetic choice*
  - *a lack of design for design's sake, to leave them set to a blue with underline*
- breadth and diversity of devices and network connected applications
  - *also see a dizzying number of evolving patterns and standards*
- consider Apple's or Google's design guidelines, then compare to Microsoft's
- no single pattern for use, no unified visual language outside of prescribed ecosystems
- want applications we design and use to be more than simply utilitarian

Design and interface - design as a guide

- interfaces simply allow us to mediate communication options and associated interaction
  - *through screens and available networks*
- definite need for a clear visual language
  - *contains signs and symbols to help inform our users*
  - *provide complementary direction and feedback*
- not as simple as just presenting the data as various forms of information

> *...primary technique to achieve improved visual communication is to use clear, distinct, consistent visible language...*
>
> *Marcus, Aaron. Graphic Design for Electronic Documents and User Interfaces*

- consider detailed, complex visual interfaces
  - *can observe the many messages conveyed on a single screen*
- challenge for design is to create some semblance of direction, order, and pattern
  - *help users simply make sense of what they see*

Design and interface - communication

- can be considered as involving
  - *a sender*
  - *a message*
  - *a conveying signal or carrier for the message*
  - *a receiver or viewer who needs to interpret the message*
- readily observe as designers and developers
  - ***we are not able to control the entire process***
- interface design
  - *the very act of selecting elements with user expectations in mind*
  - *then the combination of these elements*
  - *with appropriate and useful visual signals that users actually understand*
  - *makes it more likely a target audience of users will successfully understand and interpret our message*
- need interfaces to help us successfully manage increasingly complex nature of data

Design and interface - direction and principles

- a basic framework, a set of underlying principles we can follow or use
  - *a basic template for how we think and act as designers*
- start designing our applications with a more informed decision-making process
  - *helps us bridge form and function*
  - *provide a sense of the beautiful with the useful*
  - *such considerations are not mutually exclusive*
- underlying principles we can consider, and apply, to our designs
  - *inherently help inform good practice and design choices for our development work*
- principles will focus upon
  - ***consistency**, **hierarchy**, and basic design **personality***
- consider these underlying principles in a similar vein to syntax or language
- **Consistency** and **hierarchy** are often seen as analagous to a language's grammar
  - *a user learns whilst using an application.*
- visual **personality** of our design
  - *visual characteristics, notable traits in effect, of our design become the words we use to convey our message*
- such principles can hold true even as technology continues to evolve
- design aesthetics and principles can remain as a footprint of our work

Design and interface - tools of the trade

- consider visual tools of our trade
  - *the nuts and bolts of visual design*
- **tools** that help us layout and construct our interfaces for users
- need to define and outline the various visual tools of application design
  - *affordances*
  - *colour*
  - *controls*
  - *imagery*
  - *layout*
  - *type*

Design and interface - common mistakes

- consider some of the common mistakes
  - *affect our ability to design*
  - *implement consistency within our interfaces*
- consider interfaces that achieve consistency
  - *colours appropriate for the criteria or usage environment*
  - *consistent use of colours*
  - *consistent standards for typography*
  - *consistent implementation and styling of controls*
  - *elements correctly organised and aligned*
  - *elements placed in a logical position for users*
  - *ie: where users expect to find them*
  - *fonts used appropriate to a given situation, event...*
  - *grouping of similar, contiguous elements*

Design and interface - consistency, consistency...

- we need to establish rules for placement and usage of interface elements
- need to consistently adhere to these prescribed rules
- mix and match visual interface characteristics without confusing, and annoying, our users
- designer's visual language, like natural language
  - *requires a set of rules to be applied consistently*
  - *these rules can then be recognised and interpreted*
- consistency in design is rarely exciting or necessarily interesting
- it will help our users gain an innate sense of familiarity with an application
- hopefully helps drive further adoption and usage
- design consistency is simply about giving users what they can understand
  - *in essence, rely on throughout an application*

Image - design consistency example

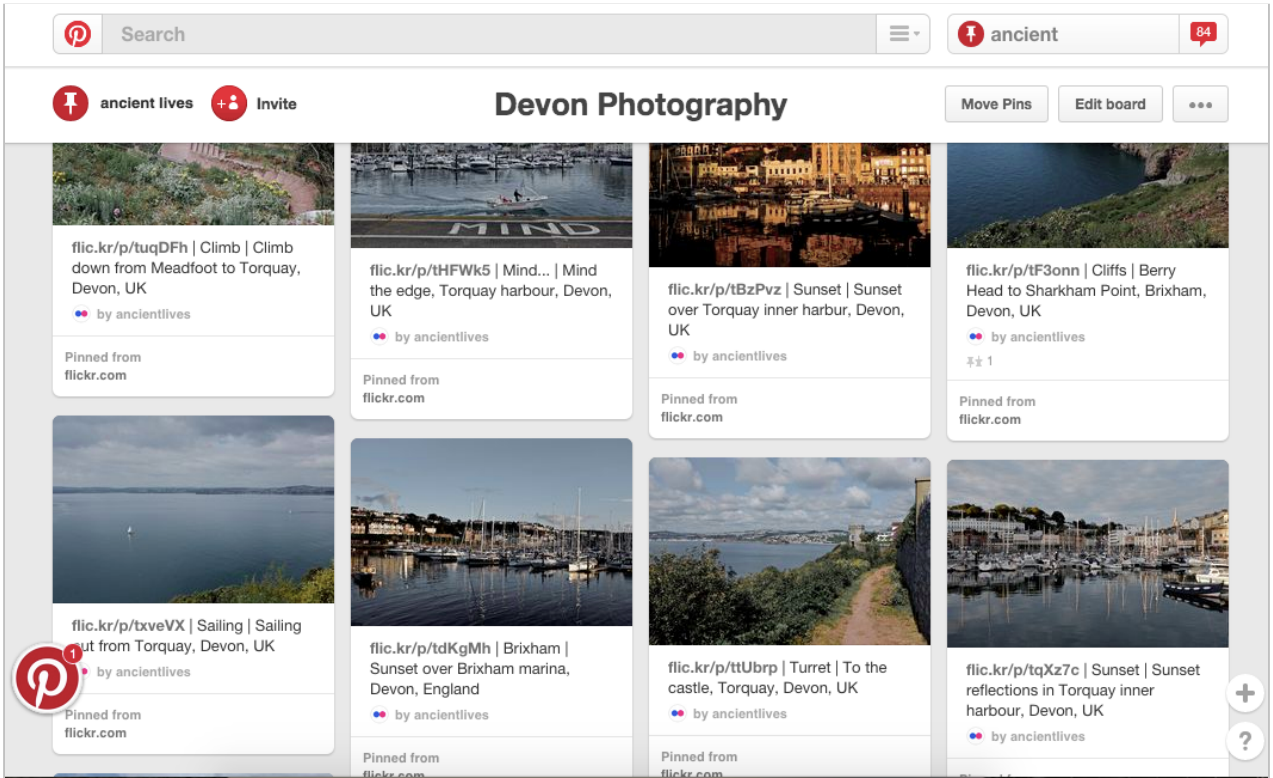
A lack of consistency - Arngren.net

Source - Arngren.net

Design and interface - considering consistency

- start to design our interfaces for applications and web sites
- then update them in response to feedback or feature changes
  - *smallest changes can cause a ripple effect throughout our application*
- applications may change and evolve, implementing new or updated technologies, options...
- still need to establish consistency in usage
  - *eg: Pinterest interface*
  - *uses an interface mechanism of continuous scrolling to display a rich variety of images*
  - *now an accepted option for an interface pattern*
- continuous scroll pattern is attempting to solve a given problem
  - *user needs to view a subset of data that is not easily displayed on a single page*
  - *application's content presented to users as focused subset*
  - *larger, seemingly endless dataset to focused view*
  - *user needs to be aware of the ongoing content*
  - *without excessive effort or hindrance to the usage experience*

Image - Pinterest continuous scroll



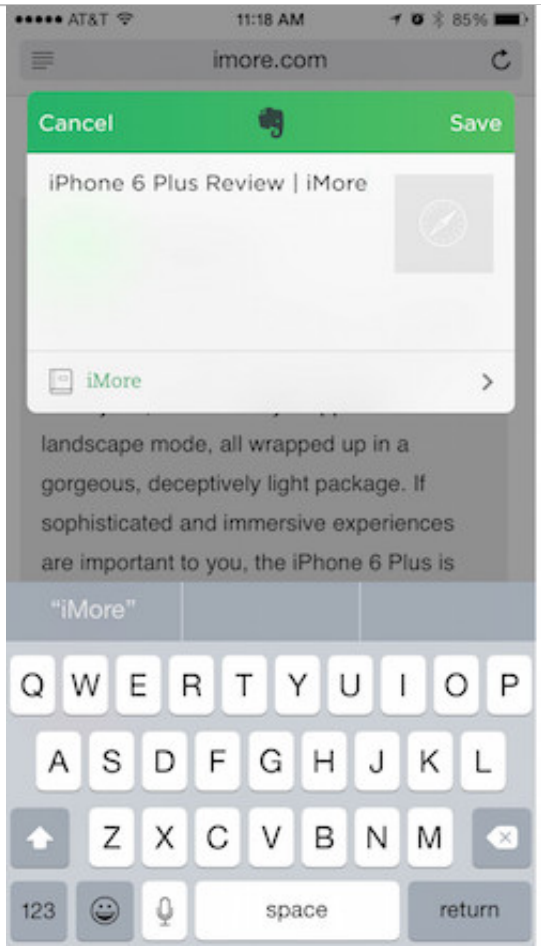
Pinterest - Continuous Scroll

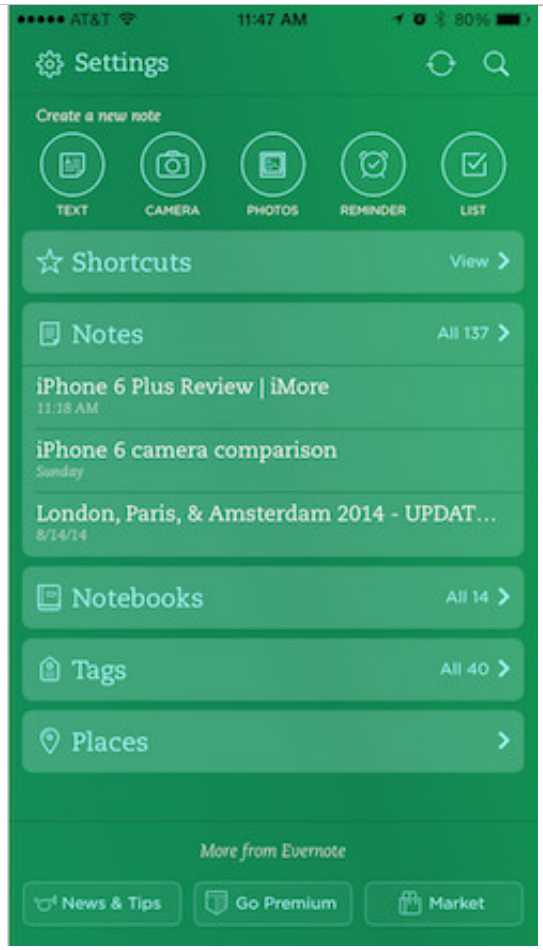
Source - Pinterest

Design and interface - establishing consistency

- help our users by starting with familiar elements and designs
  - *elements and designs people are familiar with from other examples and applications*
- users' expectations can simply be influenced by what they see onscreen
  - *naturally what they've seen in the past*
- a good reason that to review and consider many different types and examples of websites
- forms can be a good example of this type of conditioning and expectation in users
- a user sees a form for payment or credit card information
  - *they have normally seen and used other examples*
  - *examples will often follow a similar pattern*
- we can modify slightly to match specific requirements
  - *such as text, specific event or purchase details...*
- a user will normally look for familiar interface elements
  - *such as a **submit** button, input field...*
- as users, we become conditioned to use patterns on a regular basis
- consistency relies on an inherent awareness of user expectations

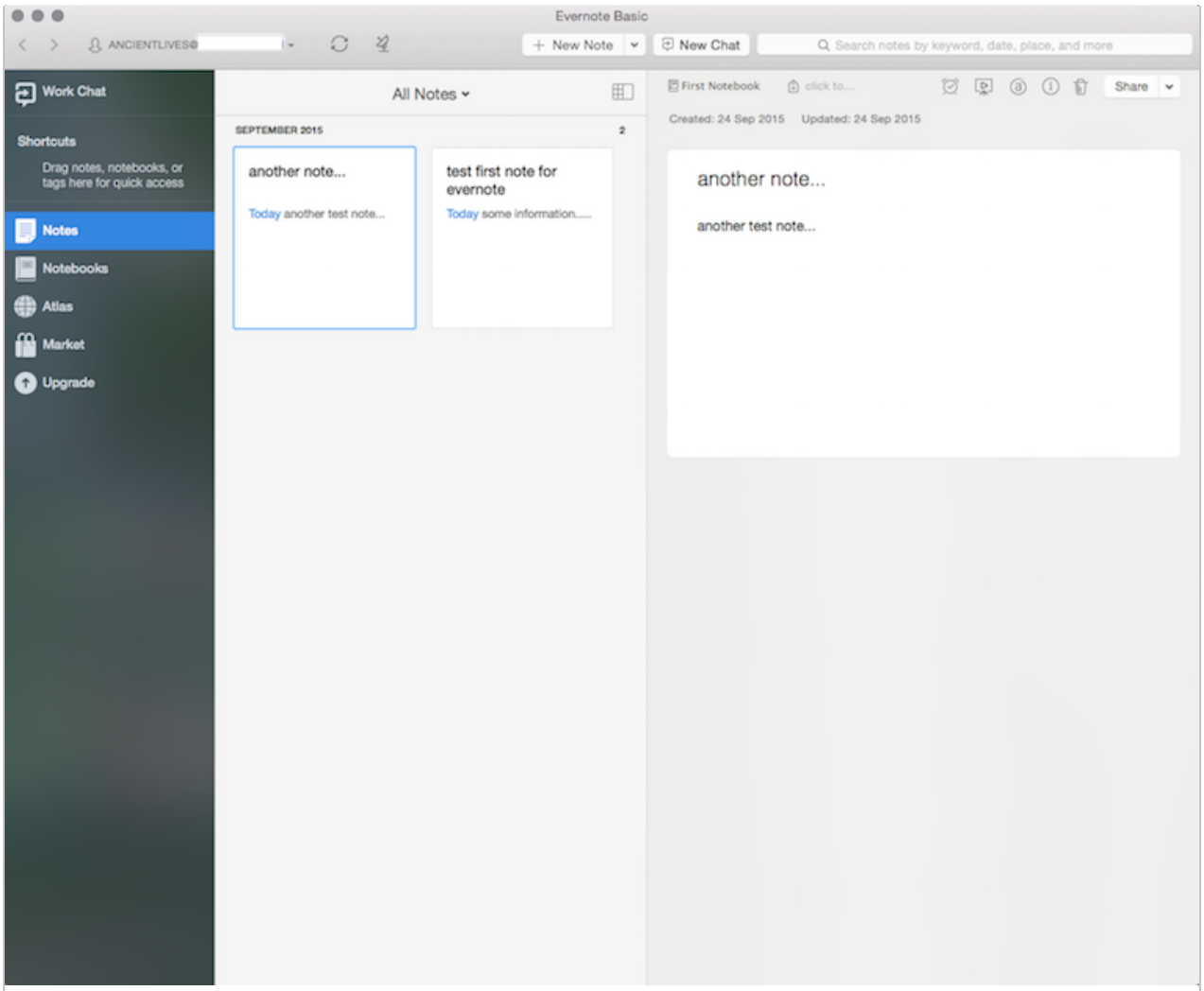
Design and interface - examples of consistency to consider...





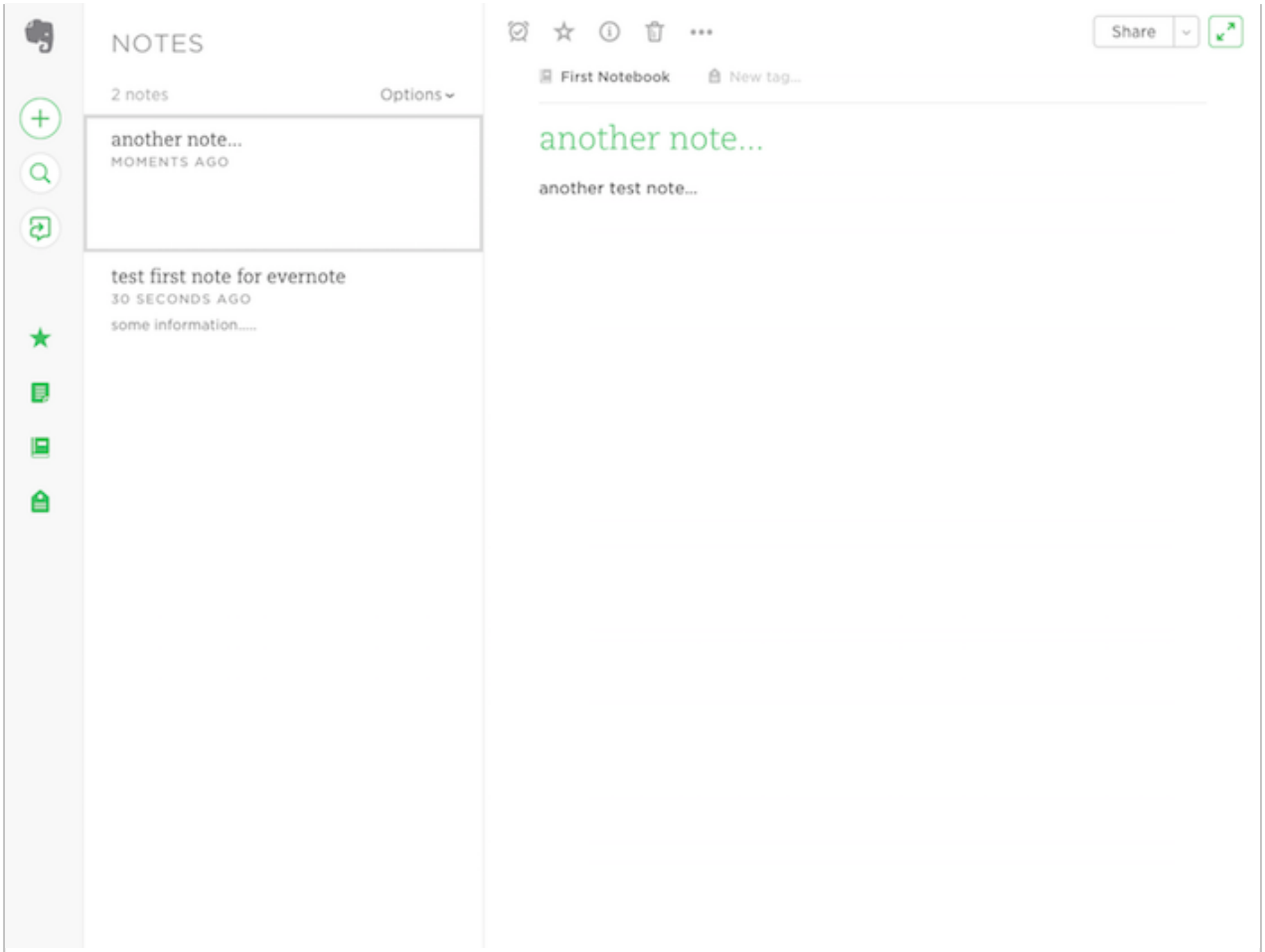
Evernote on iOS

Design and interface - examples of consistency to consider...



Evernote on OS X

Design and interface - examples of consistency to consider...



Evernote website

Demos

- JSFiddle - this - events
- JSFiddle - this - global
- JSFiddle - this - literal
- JSFiddle - this - literal 2
- JSFiddle - this - window
- JSFiddle - Parse JSON
- Travel Notes app
  - *DEMO 1 - travel notes - v0.1*
  - *DEMO 2 - travel notes - v0.2*
  - *DEMO 3 - travel notes - v0.3*
  - *DEMO 4 - travel notes - v0.4*
  - *DEMO 5 - travel notes - v0.5*
  - *DEMO 6 - travel notes - v0.6*
  - *DEMO 7 - travel notes - v0.7*
  - *DEMO 8 - travel notes - v0.8*

References

- jQuery API
- jQuery :parent selector
- JSLint - JavaScript Validator
- JSONLint - JSON Validator
- W3 - JS Object
- W3 - JS Performance