HTML, CSS, AND THE CLIENT-SIDE APP

GARANN MEANS / @GARANNM
AH, THE VIEW..
NOT A LOT OF RESPECT

lots of jQuery

where all that icky DOM stuff goes

in off-the-shelf frameworks, may be unimplemented
AT SOME POINT, THE VIEW WAS THE APP

↑ before XHRs, everything had to be synchronous
↑ one page per request
↑ pages became the core of the app
SPAS CHANGED THAT

view is just a repository

can fill with data, input fields, monitoring etc.
YET THE VIEW IS STILL THE END PRODUCT.
FAIR DEFINITION:

- presented to the user
- surface for interaction
- HTML and CSS, not just JavaScript
WE TALK ABOUT JS ALL THE TIME

 frameworks
 new language features
 HTML5 without the HTML
BOOOOOORING.
HTML AND CSS, ON THE OTHER HAND..

- the scary part!
- wrangling them is painful
- observing them is tricky
- implementations differ between browsers
- the part you can’t screw up
WE DON’T NEED TO TREAT HTML AND CSS LIKE IT’S 2002.
CONSIDER A DOT.
CONSIDER A DOT.

<div id="pt47" class="point">
  ●
</div>
CONSIDER A DOT.

```
.point {
  background-color: #B87E3D;
  height: 20px;
  width: 20px;
  border-radius: 10px;
}
```
define( [ "text!graph.tmpl" ], function( tmpl ) {
    var that = this;
    this.points = {

        "47": { 
            x: 432,
            y: 210
        },

    }

    this.render = function() {
        var el;
        for ( var pt in this.points ) {
            el = document.getElementById( "pt" + pt );
            el.style.top = that.points[ pt ].y;
            el.style.left = that.points[ pt ].x;
        }
    }
    return this;
});
WHAT WE KNOW TO BE TRUE

- HTML is content
- CSS is presentation
- they display information
- but are not information themselves
..IS WRONG.

HTML may define an object
CSS may contain data about it
either might define behavior
that’s as crucial to the app as JS
Especially when the user shows up.
ESPECIALLY WHEN THE USER SHOWS UP.
NOW HTML AND CSS ARE THE AUTHORITY ON THE DATA.
NOW HTML AND CSS ARE THE AUTHORITY ON THE DATA.
This still fits the pattern

- model supplies initial data
- view reflects data and state
- event handlers update both
BUT CONTROL HAS SHIFTED

- not encompassed by static CSS
- server has no idea what’s happening
- view responds without getting permission from the rest of the app
THIS IS THE APP BEHIND YOUR APP.
HOW WE MESS THAT UP:

- nesting needlessly
- rewriting built-in behavior
- formless blobs of markup as templates
- shotgun approach CSS
WHEN WE COULD JUST..

<div class="content">
  <div id="title-wrapper">
    <h2>a cool title</h2>
    <a href="/page1">more stuff</a>
    <a href="/page2">different stuff</a>
  </div>
  <div id="body-text">
    <p>Finally tho</p>
  </div>
</div>
WE’RE NOT SHIPPING CHINA.

- stop triple-wrapping everything
- makes things slow
- makes markup less meaningful
- resist adding elements for design objectives like padding
events: {
  "click .enter-info": "enterInfo"
},

enterInfo: function() {
  var info = this.$el.next('.info-form');
  info.show();
}
..IS UNNECESSARY

\[\text{.enter-info:target + .info-form}\]
\[\text{display: block;}\]
\]
LET HTML AND CSS DO THEIR THING

- use anchors for links
- use built-in form validation
- use lists for listing things
- try not to involve JS in visual changes
<h1>You</h1>
{{=it.username}}<br/>
{{=it.followers}}
<h2>Your news</h2>
{{~it.news :item}}
  <div class="news">
    <h3>{{=news.title}}</h3>
    by {{=news.username}}
    <p>{{=news.body}}</p>
  </div>
{{~}}
<h2>Add some news</h2>
<input type="text" id="newsTitle" />
<textarea id="newsBody"></textarea>
<input type="button" id="newsAdd" value="Add" />
You
{{#def.userInfo}}
Your news
{{#def.newsItems}}
Add some news
{{#def.newsForm}}
TIDY TEMPLATING

- partials for child objects
- partials for independent widgets
- compile and concatenate on the server
- lets markup be consistent for all uses
IT’S !IMPORTANT NOT TO DO THIS

ul {
  list-style-type: none;
  margin: 0px;
  padding: 0px;
}

#content .news p.body ul {
  list-style-type: disc;
  margin: 20px;
  padding: 20px;
}

ul.not-a-list {
  list-style-type: none!important;
  margin: 0px!important;
  padding: 0px!important;
}
NORMALIZE AND PLAN

ul.not-a-list {
  list-style-type: none;
  margin: 0px;
  padding: 0px;
}
BUILD FUNCTIONAL CSS

- no substitute for a well-maintained styleguide
- keep defaults sensible
- separate CSS and JS hooks
- accept more CSS classes as a fair price for more enhancement
OK, SO WHAT DO YOU DO?
 BETTER VIEWS

↑ make HTML strongly-typed
↑ up your template game
↑ make CSS stateful
↑ consider an “app” that’s bigger than JS application code
HTML IS THE BEST JQUERY PLUGIN EVER.
MORE THAN MEETS THE EYE:

dot
MORE THAN MEETS THE EYE:

dot tiny little application
HTML WILL ALWAYS BE UI

↑ represents data

↑ with or without content

↑ element to data is a natural relationship
BUT IT ALSO HAS BEHAVIOR

- clickable
- draggable
- editable
- form attribute functions
IT HAS RELATIONSHIPS

- nesting : namespacing
- lists : arrays or hashes
- fallbacks : overridable behavior
and proper typing

- semantic tags
- media elements
- new form element types
function Dot( opts ) {
  this.el = opts.el;
  this.x = opts.x || 20;
  this.y = opts.y || 20;
  this.update = function() {
    this.x = this.el.style.left;
    this.y = this.el.style.top;
  };
}
AND WE OVERRIDE ITS PROPERTIES

```javascript
var pt47 = new Dot(
  el: $( "#pt47" ),
  x: 432,
  y: 210
);
pt47.save = function() {
  this.update();
  $.post( "/movedot", this, ... );
};
```
Could that not become its own application?
here is my cool title!

BETTER EXAMPLE
here is my cool title!

BETTER EXAMPLE
EVEN THAT’S MORE THAN WE NEED.
<div id="title" contenteditable>
  here is my cool title!
</div>

here is my cool title!

BETTER BETTER EXAMPLE
HOW FAR IS THAT FROM AN APP?

- presents data: ✓
- switch between display and edit: ✓
- stores changes: ✓
var dirty = false;

function setDirty() {
    dirty = true;
    $(
"#blogEntry"
    ).off( "input" );
}

$(
"#blogEntry"
).on( "input", setDirty );
$(
"#blogEntry"
).on( "blur", function( e ) {
    if ( dirty ) {
        $.post( "/updateBlog", this.innerText, function() {
            dirty = false;
            $(
"#blogEntry"
    ).on( "input", setDirty );
        });
    }
});
this.dirty: false,

events: {
  "input #blogEntry": "setDirty",
  "blur #blogEntry": "storeChanges"
},

setDirty: function () {
  this.dirty = true;
},

storeChanges: function () {
  if ( this.dirty ) {
    this.model.save( { body: this.$el.text() } );
    this.dirty = false;
  }
}
WHAT WE SKIPPED

- submit button
- submit button handler
- markup and style for two modes (display and edit)
- code to switch between
INTEGRATING USEFUL HTML

- observe broadcasted events
- move current data to permanent data store
- programmatically reset
- maybe: load polyfills
- done!
USEFUL HTML BEGINS WITH USEFUL TEMPLATES.
TAKES SOME RESEARCH

↑ just use what came with your MVC framework, right?
↑ or what came with the server-side framework?
↑ BZZZZZT incorrect
SHOULD BE SPECIFIC TO YOUR APP

- are templates shared?
- do they correspond to viewmodels?
- level of granularity?
- how are they compiled and packaged?
ASSUME THE DEFAULT TEMPLATE ENGINE IS PERFECT.
FOR THE REFERENCE APP.
ALMOST ALWAYS

↑ don’t be afraid of a little logic
↑ if it’s client-side, cache it
↑ distrust magic
WRITE A RENDERER

- to manage caching and loading
- to do viewmodel type stuff
- e.g. composing data from pieces
- because having template engine API code in your view kinda sucks
function render( opts ) {
  var tmpl = tmplCache[ opts.name ];
  $.extend( true, opts.data, staticData );
  if ( tmpl ) {
    return opts.container ?
      opts.container.append( tmpl( opts.data ) ) :
      tmpl( opts.data );
  }

  if ( !tmpl && opts.url ) {
    var p = tmplCache.partials;
    $.get( opts.url, function( raw ) {
      tmpl = opts.name ?
        tmplCache[ opts.name ] = doT.template( raw, p ) :
        doT.template( raw, p );
      if ( opts.container ) {
        opts.container.append( tmpl( opts.data ) );
      }
    });
  }
}
CONSIDER REUSE

- Consider reuse on both client and server, obviously.
- Which pieces are chrome?
- Which pieces will appear multiple places?
- Do you need conditionals, or can you use CSS classes?
CONSIDER PACKAGING

- what needs to load by default
- what is lazy-loaded
- what about admin vs. normal user
- what about other languages
A template is more than the visible area of your model.
TEMPLATES INITIALIZE

- native element behaviors
- element states
- application state
ORGANIZE EVERYTHING

- templates for content
- one template minimum per widget
- templates for application states that require different markup or content
define( [ "text!item.dot", "UserView", "CatView" ],
function( itemtmpl, UserView, CatView ) {

    var ItemView = {
        init: function( container, data ) {
            tmplCache.ItemView = itemtmpl.template();
            this.container = container;
            Object.observe( data, this.render );
            // delegated event handlers go here
        },
        render: function( updates ) {
            Renderer.render( {
                name: "ItemView",
                data: updates.object
            });
        }
    };

    return ItemView;
});
template-driven

- build up dependencies in code
- partials
- CSS
- keep event handlers with the templates they select on
UPGRADE THOSE VIEWS

↑ not a reflection of a model
↑ may not have one!
↑ should wire up its own rendering
↑ therefore: its own object
↑ even if it’s a partial
CSS IS ALL THE MAGIC YOU NEED.
TYPICAL CSS, TYPICAL VIEWS

- Expect to be present already
- Switch out stylesheets for “page” changes
- Rarely tied directly to an app
WELL WHY THE HECK NOT?
AT MINIMUM, CSS IS STATE

- place in workflow
- user roles
- errors
- drilling-down
YOUR CSS MAY ALSO CONTAIN:

- data
- content
- animations
- event triggers
- application workflows
THOSE ARE APP PIECES, AND SHOULD BE MANAGED LIKE IT.
START WITH OOCSS

- your JS objects are modular
- your HTML is in templates which are in view modules
- modules can be required by a dependency manager
- GUESS WAT
WHAT GOES IN THE DEFAULT

- normalization
- layout, fonts, headers, paragraphs, etc.
- generic error states
- generic widget styling (e.g. overlays)
- composable properties
LAZY-LOAD CSS AS TEMPLATES

- for optional add-ons
- for a/b testing
- for translated content
- for interactions
YOU DON’T HAVE TO WAIT FOR JS TO CHANGE THE STATE

.high-contrast:target + .content {
  background-color: #fff;
  color: #000;
  font-size: 16px;
}

body:not(.admin) .ctrl-panel {
  display: none;
}
IN YR WIDGETS, DOING YR CLICKY THINGS

↑ again: don’t add JS to make visual changes if you don’t have to

↑ clicks that change display properties

↑ clicks that show and hide

↑ triggering animations
Using CSS behavior in your app

- In terms of JS, should be wireless
- CSS behavior often listens to the same events JS does
- Separate code in a separate file
- Thus, treat it as a module
TYING EVERYTHING TOGETHER
YOUR APP

- leans heavily on HTML and CSS
- trusts the DOM
- helps JavaScript narrow its responsibilities
- considers everything a module
YOUR APP

↑ is more awesome
THANKS!

@GARANNM / GARANN.COM

HAMMER ICON FROM THE NOUN PROJECT