Backbone Workshop

Nathaniel T. Schutta
@ntschutta

Assume you have:
browser, text editor.

Copy zip.

Or clone it from github.

```
git clone git://github.com/ntschutta/
backbone_workshop.git
```

Extract to...somewhere ;)

Shout if you have ??s

Backbone Workshop

Nathaniel T. Schutta
@ntschutta

It all started so quietly...

One line at a time.

No avoiding it now...

JavaScript is a first class citizen.

What does that mean?

How do we create modern web applications?

Applications are changing.

How do we embrace that?

How'd we get here?

In the beginning...

“I think there is a world market for maybe five computers.”

— Thomas J. Watson (attributed)

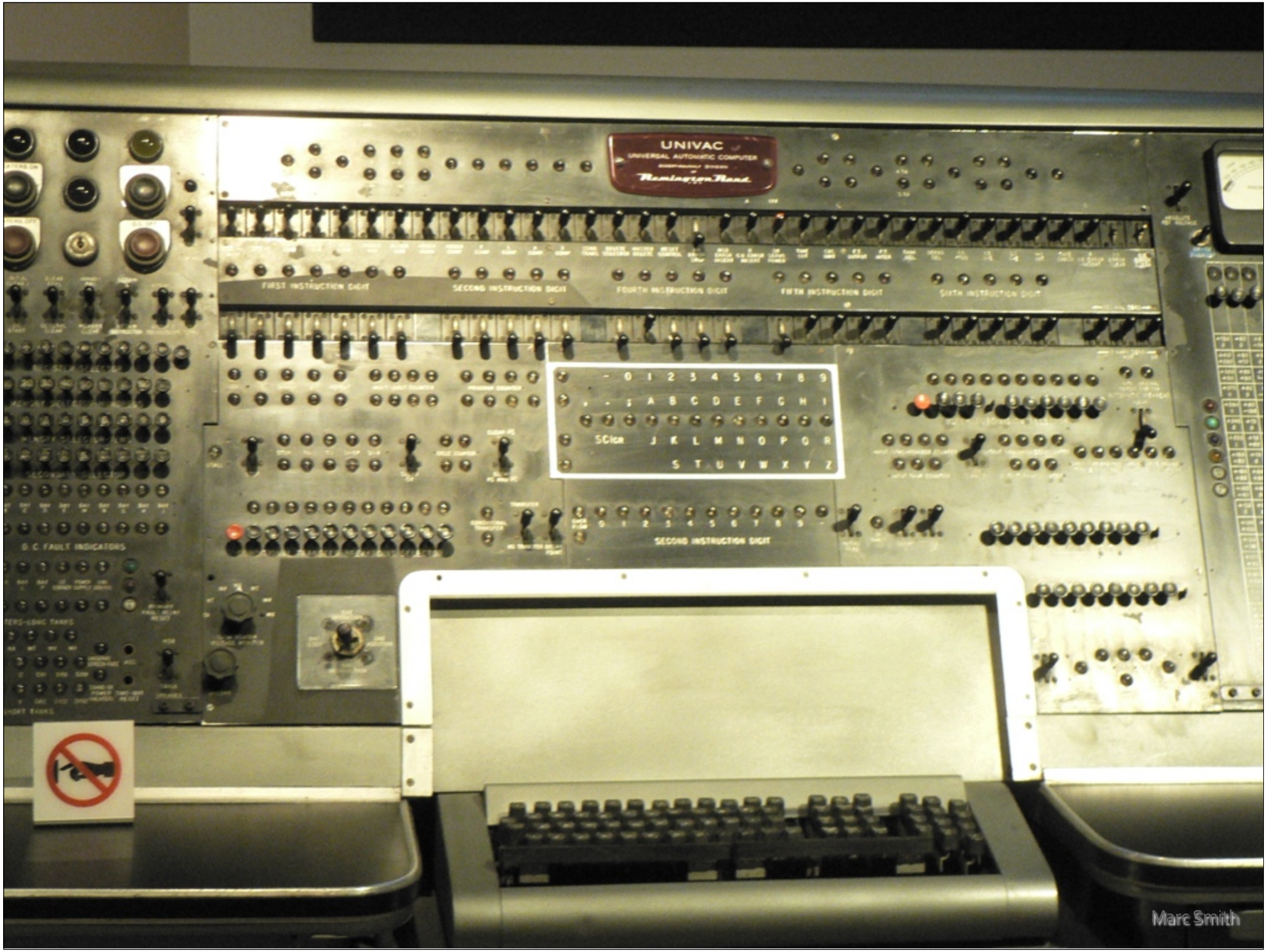
Computers were monsters.

Expensive.

Centrally managed.

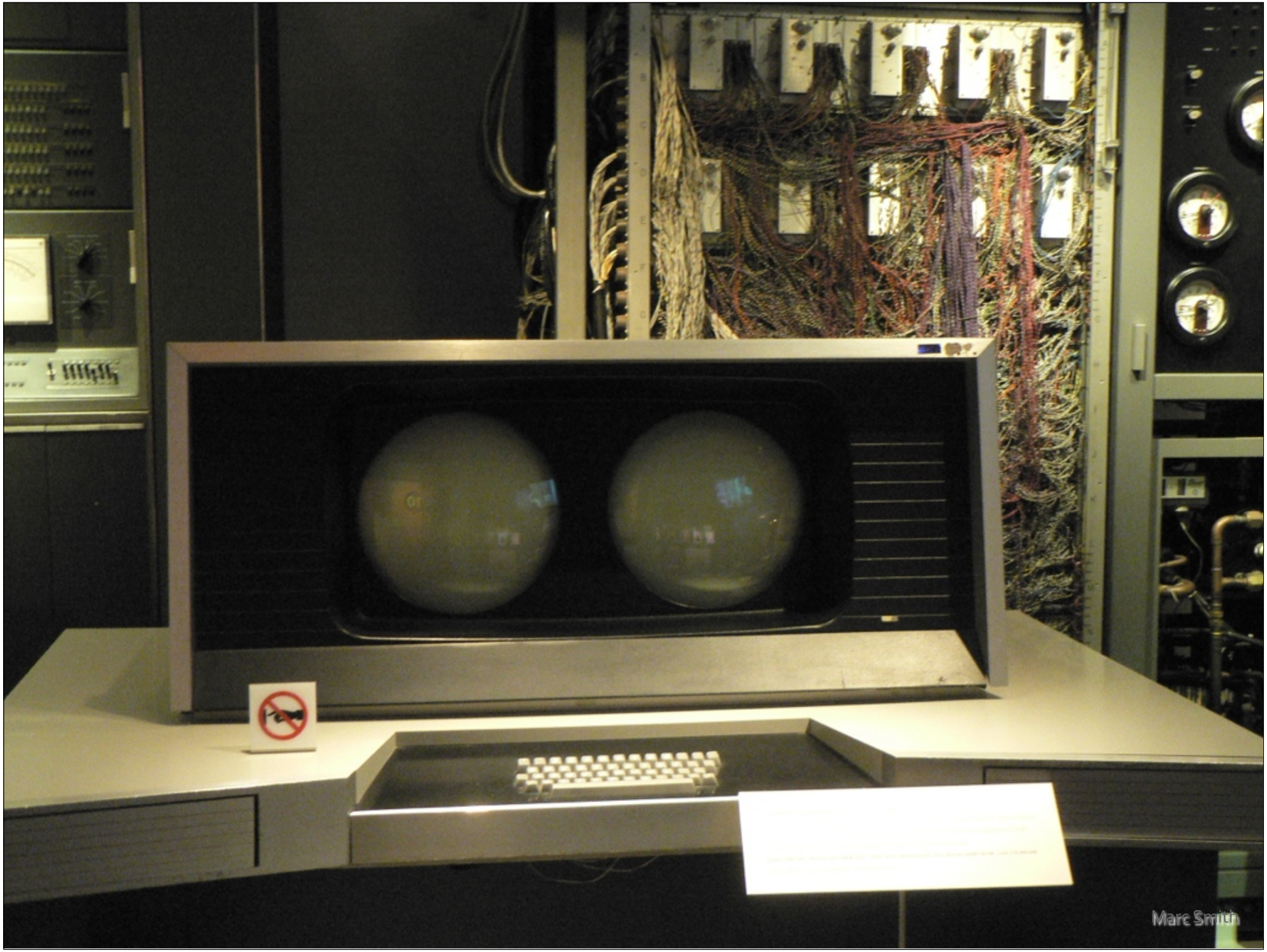
Looked like this:

IBM
SYSTEM 360


UNIVAC
UNIVERSAL AUTOMATIC COMPUTER
Remington Rand
FIRST INSTRUCTION DIGIT
SECOND INSTRUCTION DIGIT
FOURTH INSTRUCTION DIGIT
FIFTH INSTRUCTION DIGIT
SIXTH INSTRUCTION DIGIT
SECOND INSTRUCTION DIGIT
D.C. FAULT INDICATORS
Marc Smith

And we programmed using something like this:

ICL
1301


Marc Smith

Anyone remember those days?

Good times.

How did we access all that power?

encodeMessage(
checkRes
buf[i] = 0;
mess) {
buf[loc] = mess[i++];
System.exit(1000 + i);

Everyone's favorite UI,
the green screen.

The dumb terminal.

If you’ve ever traveled...

Some advantages to this model though.

Terminals were cheap.

Easy to maintain.

Didn’t have
“compatibility” issues.

But it was lacking something.

Beyond taste.

The UIs weren't exactly rich.

Not user friendly.

Eventually...

Computers got smaller.

Marc Smith

They were
dismissed as toys.

“First they ignore you, then they laugh at you, then they fight you, then you win.”

— Mahatma Gandhi (attributed)

And they weren't very powerful.

But the future was here.

And we started to create...

Client server apps.

VB, PowerBuilder, Access.

Richer UIs!

Take advantage of all that *power*.

But there were downsides.

DLLs.

Distribution.

Compatibility issues...

Ever experience that?

But, change was afoot.

In the 90s, we discovered...

The Internet.

And a revolution was born.

We created thin clients...

That, ah, talked to powerful servers.

Hmm.

Remind you of anything?

We even kept the submit button...

But they were better than a green screen right?

And, we didn't have client side issues to deal with.

Other than IE at least.

#winning

But, the UIs
weren't very rich.

Again.

Forms with holes.

Request/response paradigm.

And many people had experienced rich UIs.

Sorry.

Good for us...
less good for users.

But then what happened?

Google Maps

http://maps.google.com/maps?hl=en&tab=wl

html5 del.icio.us Saved Tabs Gmail Firebug Lite Dropbox GitHub pinboard read later Instapaper Instapaper: Read Later

+Nathaniel Search Images Maps YouTube News Gmail Documents Calendar More

Google

Nathaniel Schutta + Share

Get directions My places

Set default location

Put your business on Google Maps

Experience MapsGL

Maps Labs - Help

Google Maps - ©2012 Google - Terms of Use

Satellite

Traffic

Canada Alberta Manitoba British Columbia Saskatchewan Edmonton Calgary Winnipeg Ontario Quebec Vancouver Seattle Spokane Washington Portland Montana North Dakota Minnesota Minneapolis Wisconsin South Dakota Oregon Idaho Wyoming Salt Lake City Nevada Utah Sacramento San Francisco Stockton California Bakersfield Las Vegas Los Angeles Riverside Mexicali Tijuana Ensenada Arizona Phoenix Tucson Albuquerque New Mexico Colorado Nebraska Lincoln Omaha Iowa Chicago Illinois Michigan Madison Milwaukee Detroit Toledo Buffalo New York Cleveland Pennsylvania Philadelphia Ohio Indiana Columbus Cincinnati West Virginia Kansas City St Louis Missouri Kansas Wichita Kentucky Virginia Tulsa Oklahoma Arkansas Tennessee Memphis Atlanta North Carolina South Carolina Mississippi Dallas Alabama Mobile Georgia Texas San Antonio Louisiana Houston Jacksonville Orlando Florida Miami Hermosillo Chihuahua Ciudad Juarez Nuevo Laredo Corpus Christi Matamoros Gulf of California Ciudad Obregón Torreón Municipality Culiacán Saltillo Monterrey Durango México Ciudad Victoria La Paz Zacatecas San Luis Potosí Aguascalientes Guadalajara León Querétaro Morelia Mexico City Veracruz Oaxaca Villahermosa Acapulco Tuxtla Gutiérrez Tapachula Guatemala Honduras Nicaragua Costa Rica Panama Medellín Cúcuta Merida Cancun Campeche Cuba Dominican Republic San Juan Puerto Rico Gulf of Mexico Caribbean Sea Gulf of St Lawrence Quebec New Brunswick Ottawa Montreal Maine Nova Scotia Toronto Vermont New Hampshire Massachusetts Rhode Island Connecticut New Jersey Delaware Maryland District of Columbia United States

500 mi

500 km

Map data ©2012 Google, INEGI

And of course...

http://www.google.com/

Google

Gmail del.icio.us Firebug Lite Dropbox GitHub

Web Images Videos Maps News Shopping Gmail more ▼

iGoogle | Search settings | Sign in

Google

Advanced Search
Language Tools

Google Search I'm Feeling Lucky

Advertising Programs Business Solutions About Google

© 2010 - Privacy

Ajax.

Enter the era of RIA.

The “husky” client.

Best of both worlds?

Easy distribution, rich (enough) interfaces.

Sensing a theme here...

sylvar

Back and forth.

And we're heading there again.

Now browsers are the PC.

They're getting more powerful.

V8, Nitro, JägerMonkey.

And we're asking them to do more.

HTML5.

Web Workers, Web Sockets, Offline.

REST.

JavaScript.

First class citizen.

We're writing more of it too!

New possibilities.

And we have options.

We can break out of the request/response approach.

Click and wait?

http://alexmaccaw.co.uk/posts/async_ui

Enter asynchronous UIs.

Why?

Performance matters.

Well, perceived performance at least.

Milliseconds matter.

Amazon: 100ms delay reduces sales by 1%.

400ms on Yahoo!?

5-9% drop in traffic.

500ms extra on Google?

Drops traffic by 20%.

http://www.slideshare.net/stubbornella/designing-fast-websites-presentation

It matters.

Embraces what we're doing.

Provide structure to all that JavaScript.

I know what some of you are thinking...

Anything but JavaScript.

“Bad developers will move heaven and earth to do the wrong thing.”

— Glenn Vanderburg

Embrace it.

Partial refreshes,
JSON, services.

Takes it a step further.

MVC.

Non-blocking UI.

Decouple requests from the UI.

Render view on client.

Push state to client.

Talk to server asynchronously.

Update the UI, then tell the server about it.

Wait. What?

Things will go wrong!

Yep.

What about validation?

Server could reject the change.

Client side validation.

Need to validate on the server too...

What if the server pukes?

Error handling.

Parallel requests?

Pipeline ajax requests.

Try to navigate off gmail with update pending...

It isn’t perfect!

But there are answers to many issues.

Why should we do this?

Better user experience.

Uh no.

Some things should be synchronous!

Need *some* feedback.

Gives us another tool.

How do we do it?

The server.

REST.

FTW.

Not request/response...

Finer grained.

May need support
for web sockets.

Jetty, Node, Socket.IO...

The client.

State and view.

Preload data.

Server communication is asynchronous.

Update the client then tell the server what happened.

Opposite of what we've done for years.

Hmm, managing state on the client sounds hard.

Can be.

JavaScript is often...
lacking in structure.

Probably want to
use a library!

Typically built around MVC or MVP.

Backbone.js, Spine.js, Sammy.js, KnockoutJS...

List grows daily.

How do I know
which one to use?

Play with them.

Compare them.

https://github.com/addyosmani/todomvc

“The Seven Frameworks.”

http://blog.stevensanderson.com/2012/08/01/rich-javascript-applications-the-seven-frameworks-throne-of-js-2012/

Backbone.js

Very lightweight.

As in ~5 KB compressed.

~1300 lines.

Fully documented.

Isn’t a “UI” framework.

If you need widgets, go to YUI, jQuery UI, etc.

Built for MVC
JavaScript applications.

Models, events, collections, views.

Controllers, persistence.

Influenced by Ruby on Rails.

Data lives in models.

Not the DOM.

Changes to models trigger change events.

Views are notified of said changes to the model.

Update accordingly.

No more find stuff and change it - it just updates.

You'll be coding to events.

Let's look at some code.

Backbone.Model

Create models that extend Backbone.Model.

Add properties and methods.

Your models inherit a ton of behavior.

- get/set
- has
- clear
- toJSON
- save
- validate
- clone
- changedAttributes
- previous
- ...

And much more!

Provides an empty validate method.

You provide implementation.

set() and save() halt on invalid data.

Provides a way of setting default values.

Lab time!

Model Basics

- Build a todo model (there's a shell waiting for you)
- Extend Backbone's model
- Add default attributes
- ${extract}/backbone_workshop/labs/js/models/todo.js
- solution: ${extract}/backbone_workshop/labs_finished/js/models/todo_basics.js

Backbone.Collection

Includes some fancy collection magic.

Sets of models.

Usually of a single model type.

Events fire when items in the collection change.

Also when items are added or removed.

Borrows from
Underscore.js as well.

You may also see references to Lo-Dash.

https://github.com/bestiejs/lodash

Gain some nifty iteration functions.

- add/remove
- get
- sort
- pluck
- parse
- fetch

And more.

Retrieve models via client IDs or model's ID.

Collections can be ordered.

Provides a richer comparotor concept.

Also adds a *fetch* to retrieve collections from server.

Provide a URL endpoint.

Backbone.View

Convention.

Not templates.

Often used with a template library.

Such as Mustache.js,
Haml-js, or Eco.

Handle presentation.

Linked to a DOM element.

this.el

Can bind directly to an existing element.

Defaults to an empty div.

Bind a view's render object to the change in a model.

Instead of a series of queries and DOM updates.

Extend Backbone.View.

Implement render.

Return the right HTML.

Update el with said HTML.

Again, probably using a template library.

Model has `toJSON()` to feed data to template.

Also gives an event hash.

Easy way to bind to interesting events.

{"eventType selector":
"callback"}

Selector is optional.

Leave it off? Binds to *el*.

Lab time!

View Basics

- Extend Backbone's view
- Add a click handler to toggle the state of the todo item
- Add a click handler for removing a todo item
- Add a listener to trigger the render function when the todo model changes
- ${extract}/backbone_workshop/labs/js/views/todos.js
- solution: ${extract}/backbone_workshop/labs_finished/js/views/todos_basics.js

And so much more...

Putting it all together.

Several sample apps.

todos.js

http://documentcloud.github.com/backbone/docs/todos.html

Backbone Demo: Todos

http://documentcloud.github.com/backbone/examples/todos/index.html

Todos

What needs to be done?

Double-click to edit a todo.
View the annotated source.

Created by
Jérôme Gravel-Niquet

```
window.Todo = Backbone.Model.extend({

    defaults: function() {
      return {
        done:  false,
        order: Todos.nextOrder()
      };
    },

    toggle: function() {
      this.save({done: !this.get("done")});
    }

  });
```

```
window.TodoList = Backbone.Collection.extend({

    model: Todo,

    localStorage: new Store("todos"),

    done: function() {
      return this.filter(function(todo){ return todo.get('done'); });
    },
```

```
window.TodoView = Backbone.View.extend({

    tagName:  "li",

    template: _.template($('#item-template').html()),

    events: {
      "click .check"              : "toggleDone",
      "dblclick div.todo-text"    : "edit",
      "click span.todo-destroy"   : "clear",
      "keypress .todo-input"      : "updateOnEnter"
    },

    initialize: function() {
      this.model.bind('change', this.render, this);
      this.model.bind('destroy', this.remove, this);
    },

    render: function() {
      $(this.el).html(this.template(this.model.toJSON()));
      this.setText();
      return this;
    },
```

Works like this:

Backbone Demo: Todos

http://documentcloud.github.com/backbone/examples/todos/index.html

Google

html5 del.icio.us Saved Tabs Gmail Firebug Lite Dropbox GitHub pinboard read later Instapaper Instapaper: Read Later

Todos

What needs to be done?

Double-click to edit a todo.
View the annotated source.

Created by
Jérôme Gravel-Niquet

Also an “app” level view.

Notice it uses some jQuery!

Not so hard!

Wine Cellar.

http://coenraets.org/blog/2011/12/backbone-js-wine-cellar-tutorial-part-1-getting-started/

Let's extend the basics.

We're going to add a priority button...

In small steps.

What should this do?

Allow us to say a todo item is high, medium, low.

First, we need some buttons.

Where does that go?

The CSS is already there.

Lab time!

Priority Buttons

- Add three buttons for priority - High, Medium, Low
- ${extract}/backbone_workshop/labs/index.html
- solution: ${extract}/backbone_workshop/labs_finished/index_basic.html

OK, now we want those buttons to *do* something.

Let's just alert the priority of the clicked button.

Where would we go to add an event handler?

How do we add an event handler?

Lab time!

Wire up the Buttons

- Add a handler that calls a method when a priority button is clicked
- Implement the method to alert the priority of the button that was clicked
- ${extract}/backbone_workshop/labs/js/views/todos.js
- solution: ${extract}/backbone_workshop/labs_finished/js/views/todos_wire_up.js

OK, we know the buttons work when clicked.

But an alert isn’t user friendly.

Let's change the CSS based on the priority.

Where was that event handler again?

How do we update CSS?

jQuery to the rescue!

jQuery can remove and add classes.

<cough>

Lab time!

Change the View

- Modify the priority button handler to change the CSS of the todo item based on the priority
- ${extract}/backbone_workshop/labs/js/views/todos.js
- solution: ${extract}/backbone_workshop/labs_finished/js/views/todos_update_css.js

OK, we updated the CSS.

But that should change the model too right?

Where do we go to modify a model?

Where would we actually update the model value?

How would we change a model's attribute?

How do we persist the model?

What's special about those methods?

Lab time!

Update the Model

- Add a default priority attribute to the model
- In the todo item view, update the priority handler to set the current priority on the model, save the model
- ${extract}/backbone_workshop/labs/js/models/todo.js
- ${extract}/backbone_workshop/labs/js/views/todos.js
- solution: ${extract}/backbone_workshop/labs_finished/js/models/todo.js
- solution: ${extract}/backbone_workshop/labs_finished/js/views/todos_update_model.js

We're almost there...

We've changed the view and updated the model.

But what happens when we refresh the page?

Where do we update the view of the model?

What do we have to change?

Lab time!

Display the Priority

- Update the render method to display the todo item's priority
- ${extract}/backbone_workshop/labs/js/views/todos.js
- solution: ${extract}/backbone_workshop/labs_finished/js/views/todos_display_priority.js

Now that we have priority...

Let's do something with it.

How many high priority items do we have?

Where do we start?

What do we need?

How do we get a count?

Lab time!

Display the Priority

- Update the page to show a count of the high priority items
- Add a filter to the collection to return the high priority items
- Update the application to show the count of high priority items

Display the Priority

- ${extract}/backbone_workshop/labs/index.html
- ${extract}/backbone_workshop/labs/js/collections/todos.js
- ${extract}/backbone_workshop/labs/js/views/app.js
- solution: ${extract}/backbone_workshop/labs_finished/index_high_count.html
- solution: ${extract}/backbone_workshop/labs_finished/js/collections/todos.js
- solution: ${extract}/backbone_workshop/labs_finished/js/views/app_high_count.js

The app has some filters.

How do they work?

How do we add to it?

Let's create a filter for the high priority items.

How do we hide items?

Where might that code be?

Lab time!

Display the Priority

- Add a "high" filter link to the footer area
- Update the view to filter out the non high priority items when the high filter is invoked
- ${extract}/backbone_workshop/labs/js/views/todos.js
- ${extract}/backbone_workshop/labs/index.html
- solution: ${extract}/backbone_workshop/labs_finished/js/views/todos.js
- solution: ${extract}/backbone_workshop/labs_finished/index.html

What else?

Backbone.Router

The controller.

Web apps should be linkable & bookmarkable.

Backbone.Router helps.

Connects state to URL hashes.

History API can handle much of what we'd want.

Backbone fills in where browsers fall down.

Connects and routes pages.

Shocking.

- routes
- navigate

Works in conjunction with Backbone.history.

saveLocation()

Backbone.sync

Model updates need to get to the server.

When a model changes, Backbone informs server.

By default, makes a RESTful JSON request.

sync(method, model, [options])

If call succeeds, client side model is updated.

Based on jQuery.

method - CRUD.

CRUD	HTTP
create	POST
read	GET
update	PUT
delete	DELETE

model - the thing
that changed.

options - additional callbacks, ajax options.

Expects server to return updated attributes as JSON.

Save is asynchronous.

Free to bind to any of the ajax callbacks.

Can override to use local storage, WebSockets, etc.

There's an existing local storage adapter.

https://github.com/jeromegn/Backbone.localStorage

The server side.

Expects certain RESTful endpoints to exist.

POST	/collection
GET	/collection
GET	/collection/id
PUT	/collection/id
DELETE	/collection/id

Backbone handles model serialization.

Endpoints should return model as JSON.

But we use jQuery.

You bet!

How many of you use jQuery today?

So do we.

Rocking good library.

Bob Jagendorf

Some would say it's a bit heavyweight.

Like, say, the jQuery core team.

https://groups.google.com/forum/#!topic/jquery-bugs-team/17rGK6eAAxI/discussion

http://blog.jquery.com/2011/11/08/building-a-slimmer-jquery/

1.7 deprecated a lot.

Are you using all of it?

Probably not.

Dragging along a lot of excess baggage.

Is that an issue?

It depends.

For desktop users?
Probably not.

But what about mobile?

Extra bits matter.

Especially on certain cell networks...

Movement today towards micro frameworks.

Do one or two things...

Really well.

Unix model!

Small, loosely coupled.

Pros:

If you're not using it, you don't need to push it.

Smaller learning curve.

Constraints shall set you free.

Cons:

May need multiple libraries.

Reinvent the wheel.

Existing skill sets.

Three rough categories.

jQuery alternatives.

JavaScript MVC.

JavaScript alternatives.

Some libraries
build on jQuery.

Does anyone use Backbone?

Yes!

LinkedIn Mobile,
DocumentCloud, Flow.

Foursquare, Khan Academy, Do, Posterous.

Groupon, Basecamp
Mobile, Stripe, Pandora.

Soundcloud, Code School, SeatGeek, Kicksend.

Decide, Trello, QuietWrite.

More and more.

Your competitors?

They won't tell you why they're beating you.

PAZI - MINE
ПАЗИ - МИНЕ
UKLANJANJE OVOG ZNAKA
KRIVIČNO JE DJELO
УКЛАЊАЊЕ ОВОГ ЗНАКА
КРИВИЧНО ЈЕ ДЈЕЛО

This isn’t for everyone.

You will write JavaScript.

Sorry.

Requires a rethinking of your application.

Probably can't "port".

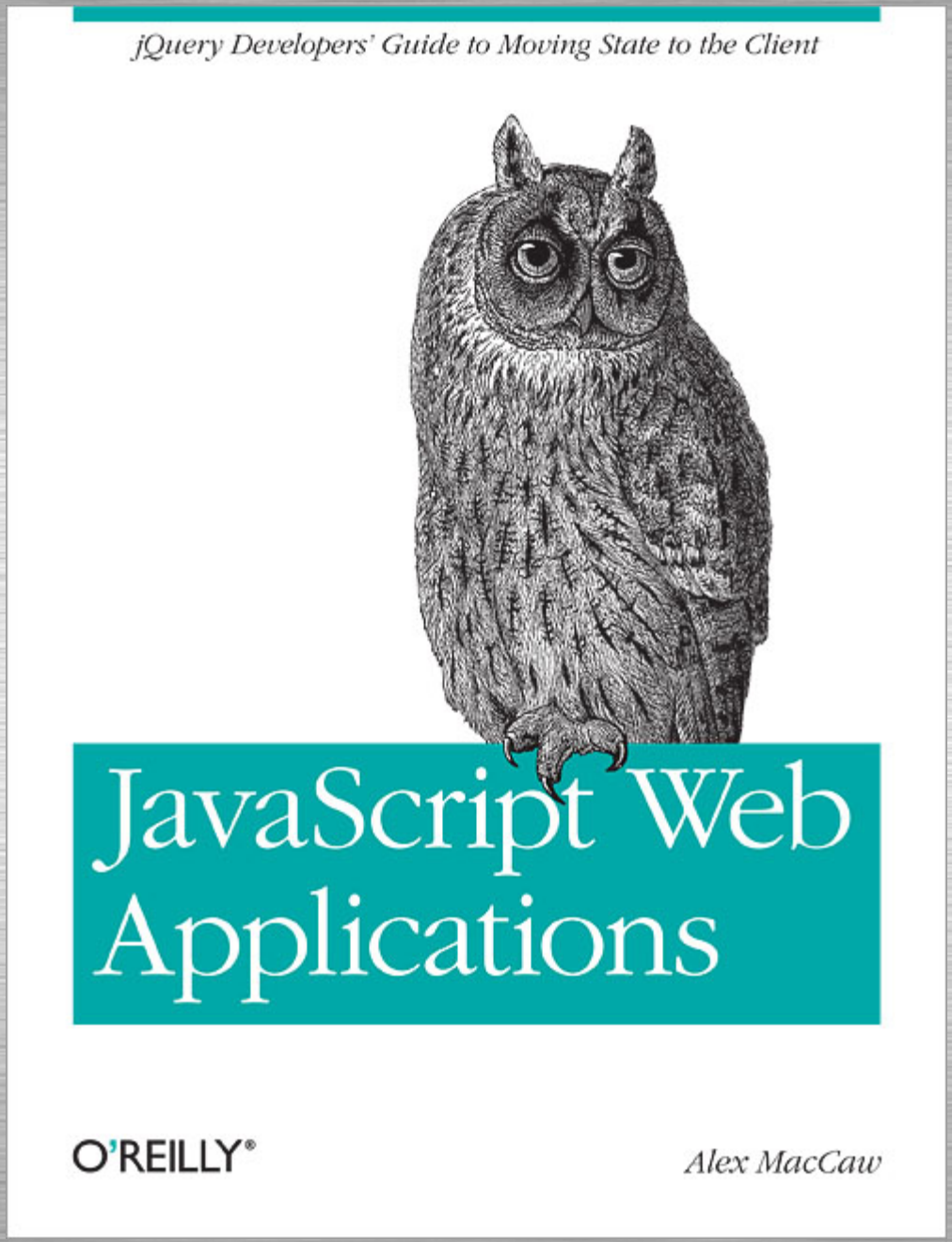
It is different.

It is new.

As in 0.9.2.

Evolving.

JavaScript Web Applications.



http://shop.oreilly.com/product/0636920018421.do#

Start thinking about it.

Where would it fit for you?

It can be done!

Be aware of the alternatives.

What are they good for?

What shouldn't they be used for?

How might they fit in your world?

Image Credits

- http://www.flickr.com/photos/marc_smith/6246956530/
- http://www.flickr.com/photos/marc_smith/6246433641/
- http://www.flickr.com/photos/marc_smith/6246957472/
- http://www.flickr.com/photos/sylvar/70589378/
- http://www.flickr.com/photos/bobjagendorf/5492860578/

Thanks!

Nathaniel T. Schutta
@ntschutta