

JavaScript Framework Face off

Nathaniel T. Schutta
[@ntschutta](https://twitter.com/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?

JavaScript MV*

APP development
is like fashion.

We go back and forth.

We've gone from dumb
terminals to thick clients...

To web pages.

Are web apps any different
than green screens?

Forms with holes.

Request/response paradigm.

The web has images
and can scroll!

But it is the same concept.

The pendulum
keeps swinging.

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://alexmaccaaw.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%.

4ooms on Yahoo!?

5-9% drop in traffic.

500ms extra on Google?

Drops traffic by 20%.

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

Or MVVM.

Or MVP.

MV⁺, MV-whatever.

Same basic approach we've
used on the server...

But now on the client.

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 again?

JavaScript isn't just for
interns anymore.

We need a way to
manage our JavaScript.

Mobility matters.

These libraries are
generally lightweight.

Drives a clean
separation of concerns.

Server basically JSON pump.

Variety of clients consume
the same services.

Certain amount of
future proofing.

What client technology
will dominate in 2 years?

Probably won't be
like today.

The client figures out the best way to present data.

To a certain extent,
this is our fault.

Web apps were simpler
“back in the day”.

We kept pushing boundaries.

Customers expect more
from a web app today.

Better user experience.

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,
Tomcat, JBoss, GlassFish.

Nearly every language has support now.

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.

<http://todomvc.com>

“The Seven Frameworks.”

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

In general, a lot of
common ground.

Progressive enhancement
isn't the answer.

Model/view
separation is key.

Not what we call it.

Declarative binding is hip.

Stick a fork in IE 6.

Many don't support IE 6,
some require 9 or greater.

MIT license is very popular.

Most live on GitHub.

Some areas of contention.

Library vs. Framework.

Somewhat semantic.

In essence:

Libraries work with your
existing architecture.

Frameworks come with
their own structure.

Ember is by far the most
opinionated in this regard.

Mandatory vs. optional.

Some enforce specific
view, storage or routing.

Others let you pick
whatever you want.

How should
templates work?

Handlebars is very
popular in this space.

String vs. DOM based.

DOM based may be a
future browser feature.

Some have specific server
expectations (Rails, etc).

Others are agnostic.

Lots of options -
paradox of choice!

Backbone.js

Very lightweight.

As in ~7.3 kB compressed.

~1900 lines.

Fully documented.

Isn't a “UI” framework.

If you need widgets, go
to 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.

Code structure.

A screenshot of a Mac OS X Finder window displaying the file structure for a Backbone.js application named "backbone_workshop". The window shows a tree view of files and folders under the "labs" directory.

The file structure is as follows:

- assets
- index.html
- js (selected folder)
 - app.js
 - collections (folder)
 - todos.js
 - lib (folder)
 - models (folder)
 - todo.js
 - routers (folder)
 - router.js
 - views (folder)
 - app.js
 - todos.js

todos.js

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

Todos

What needs to be done?

Double-click to edit a todo.

[View the annotated source.](#)

Created by

[Jérôme Gravel-Niquet](#)

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

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.

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.

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.*

Very minimalist.

Easy to adopt.

Doesn't take over your
entire application.

MIT License.

Current version: 1.2.3.

<http://backbonejs.org>

Knockout.

Model View View Model.

Smallish.

22 kb minified, gzipped.

Large compared to some!

No dependencies.

Supports mainstream
browsers including IE.

Similar concepts: models,
changes update view.

Declarative bindings.

Includes templating.

Interactive tutorial.

<http://learn.knockoutjs.com/#/?tutorial=intro>

learn.knockoutjs.com

Tutorial: Introduction | ▾

help main site

Step 1 of 5

Welcome!

In this first tutorial you'll experience some of the basics of building a web UI with the *Model-View-ViewModel* (MVVM) pattern using knockout.js.

You'll learn how to define a UI's appearance using **views** and **declarative bindings**, its data and behavior using **viewmodels** and **observables**, and how everything stays in sync automatically thanks to Knockout's **dependency tracking** (even with arbitrary cascading chains of data).

Output

First name: todo

Last name: todo

Run (Ctrl+Enter)

```
<!-- This is a *view* - HTML markup that defines the appearance -->
<p>First name: <strong>todo</strong></p>
<p>Last name: <strong>todo</strong></p>
```

```
// This is a simple *viewmodel* - JavaScript that defines the behavior
function AppViewModel() {
    this.firstName = "Bert";
    this.lastName = "Bertington";
}

// Activates knockout.js
ko.applyBindings(new AppViewModel());
```

```
<p>First name: <input data-bind="value: firstName" /></p>
<p>Last name: <input data-bind="value: lastName" /></p>

<p>Full name: <strong data-bind="text: fullName"></strong></p>

<button data-bind="click: capitalizeLastName">Go caps</button>
```

```
function AppViewModel() {
    this.firstName = ko.observable("Bert");
    this.lastName = ko.observable("Bertington");

    this.fullName = ko.computed(function() {
        return this.firstName() + " " + this.lastName();
    }, this);

    this.capitalizeLastName = function() {
        var currentVal = this.lastName();
        this.lastName(currentVal.toUpperCase());
    };
}
```

Higher level.

Links UI and model.

Can have multiple bindings.

```
<input id="new-todo" data-bind="value: current, valueUpdate: 'afterkeydown', enterKey: add">
```

```
var Todo = function (title, completed) {
    this.title = ko.observable(title);
    this.completed = ko.observable(completed);
    this.editing = ko.observable(false);
};

self.add = function () {
    var current = self.current().trim();
    if (current) {
        self.todos.push(new Todo(current));
        self.current('');
    }
};
```

Bit of an MS flavor to it.

Built by an MS employee.

Supported in Visual Studio.

Works with jQuery.

Combines with other
libraries as needed.

No forced architecture.

Doesn't take over your
entire application.

MIT License.

Current version: 3.4.0.

<http://knockoutjs.com/>

Angular.

MV-whatever.

Developed by Google.

Used in production by
Google today.

Trying to make up for lack
of app support in HTML.

A shim until browsers
reach their full potential.

Declarative binding.

Dependency injection.

Custom attributes: ng-.

Data binding updates view
or model automatically.

Controllers abstract
away the DOM.

Angular injects services
into controllers.

Supports linking,
bookmarking.

Includes form validation.

XHR wrapper: promises
and exception handling.

Supports creating
components via directives.

Supports localization.

As usual, a todo app...

```
<section id="todoapp" ng-controller="TodoCtrl">
  <header id="header">
    <h1>todos</h1>
    <form id="todo-form" ng-submit="addTodo()">
      <input id="new-todo" placeholder="What needs to be done?"
        ng-model="newTodo" autofocus>
    </form>
  </header>
```

```
todomvc.controller('TodoCtrl', function TodoCtrl($scope, $location, todoStorage, filterFilter) {
  var todos = $scope.todos = todoStorage.get();

  $scope.newTodo = '';
  $scope.editedTodo = null;

  $scope.$watch('todos', function () {
    $scope.remainingCount = filterFilter(todos, {completed: false}).length;
    $scope.completedCount = todos.length - $scope.remainingCount;
    $scope.allChecked = !$scope.remainingCount;
    todoStorage.put(todos);
  }, true);

  if ($location.path() === '') {
    $location.path('/');
  }

  $scope.addTodo = function () {
    if (!$scope.newTodo.length) {
      return;
    }

    todos.push({
      title: $scope.newTodo,
      completed: false
    });

    $scope.newTodo = '';
  };

  $scope.editTodo = function (todo) {
    $scope.editedTodo = todo;
  };
});
```

Strong focus on testing.

Has its own mocks.

Includes a scenario runner.

Jasmine plugin.

Chrome plugin.

Doesn't enforce a specific
server architecture.

Doesn't need to own the
entire application.

Works with other libraries.

Good documentation.

Tutorials and examples.

Isn't just used by Google.

MIT License.

Current version: 1.5.0.

What about 2.0?!?

Yes 2.0 will likely be an interesting upgrade...

Maybe there will be an
upgrade tool. Maybe not.

Need to ship today? Use the
latest production version.

Keep a weather eye out.

Things can and do change.

At least parts of it are
back ported from 2.

May be able to run both 1.x
and 2.x in the same app.

But yes, 2 will introduce
breaking changes.

And you can start
playing with it today.

<https://angular.io>

<http://angularjs.org>

Is that all of them?

Not even close.

Active space.

Expansion phase.

Wide variety to
choose from.

Node.js

Express + gcloud-
node

Compare these to a non-framework implementation

Vanilla JS

jQuery

“TodoMVC is an immensely valuable attempt at a difficult problem - providing a structured way of comparing JS libraries and frameworks. TodoMVC is a lone data point in a sea of conjecture and opinion.”



Justin Meyer

Selecting a Framework

Once you've downloaded the latest release and played around with the apps, you'll want to decide on a specific framework to try out.

See the [FAQ](#) for more information.

New in 1.3

- ✓ We now have 64 applications.

New since 1.3 ▾

- ✓ Updates have been made to apps including Vanilla, Angular, React.

Getting Involved

Is there a bug we haven't fixed or an MV* framework you feel would benefit from being included in TodoMVC?

If so, feel free to fork the repo, read our

What needs to be done?

- redesign website
- do some nerdy stuff

1 item left

All

Active

Completed

Clear completed

Paradox of choice!

Many approaches, same
basic concepts.

Similar influences.

which looks right to you?

Differences largely in view.

And how it wires in.

Some are more...
opinionated than others.

Some are very simple.

Use a little here,
a little there.

Others are more “all in”.

Impacts file structure,
templating, etc.

The learning curve
varies widely.

Hello World is easier in
some than others.

Some (seem) to have
smoother “complexity curves”.

Many of these
are quite new.

What is your
appetite for “new”?

Browser support.

Varies.

Some leave IE behind.

Will that work for you?

What does it depend on?

Read the docu.

Most of it is quite good.

How much should the
library do for you?

Greenfield or existing app?

Choose your own
adventure book?

Would you prefer “this is
the one true way”?

Lot of options.

Play with them.

Does anyone use
JavaScript MVC?

Yes!

More and more.

Your competitors?

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

I can't wait to build a
JavaScript MVC app!

PAŽI - 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".

Do you have a robust set
of RESTful web services?

Do you have a team of
experienced JS devs?

It is different.

It is new.

Evolving.

JavaScript Web Applications.

jQuery Developers' Guide to Moving State to the Client



JavaScript Web Applications

O'REILLY®

Alex MacCaw

<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?

Thanks!

Nathaniel T. Schutta
@ntschutta