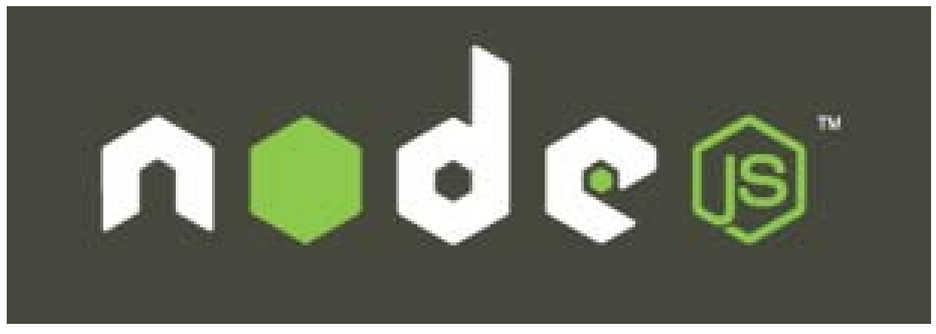
Javascript applications and the quest for Sanity

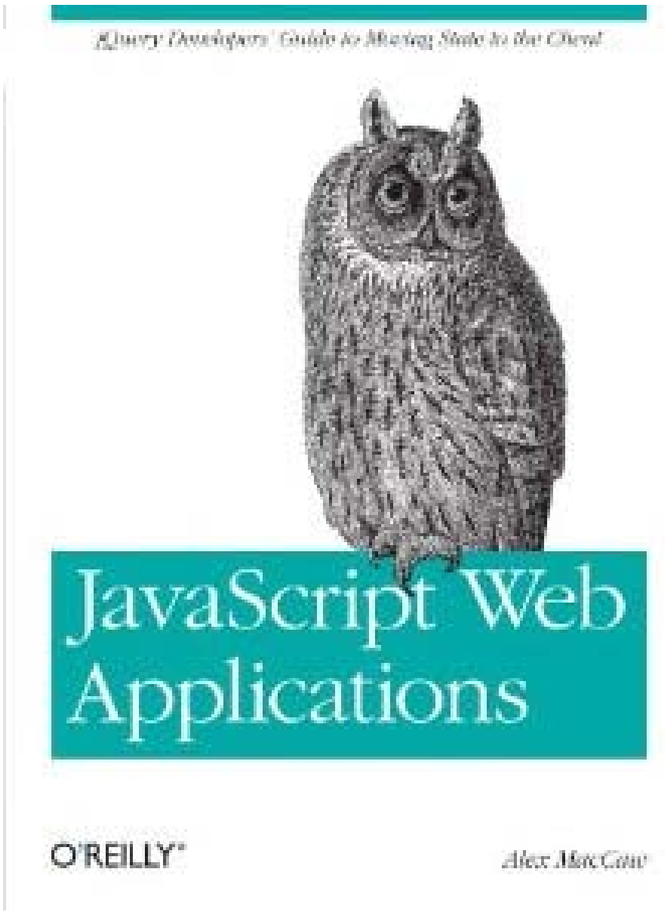
Facts about Javascript

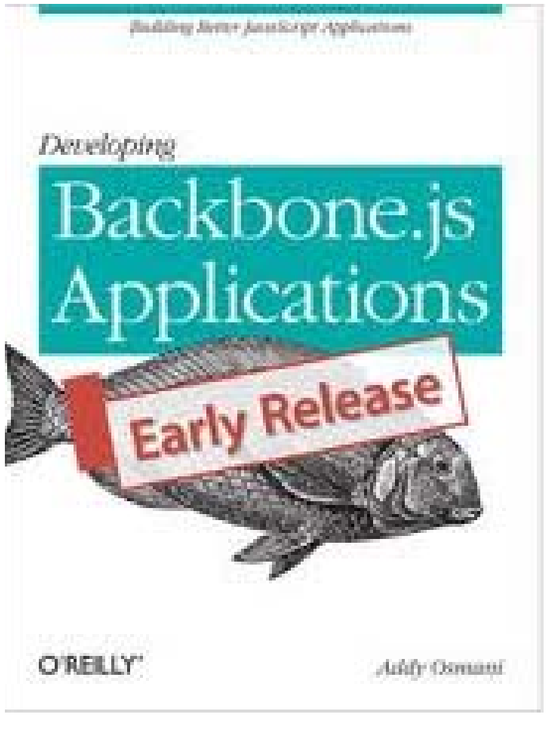
Javascript was spawned and mutated (not born and nurtured) in the dank halls of Netscape tor.

Peace across the land



Modern Javascript is culture





The Good Parts

Javascript is a flexible, loosley typed language.

It can be and has been bent in many different ways.
eg check out http://dailyjs.com/2012/11/26/js101-proto/

```
function User() {
}

User.prototype.greet = function() {
  return 'hello';
};

var user = new User();
user.greet();
```

The Good Parts

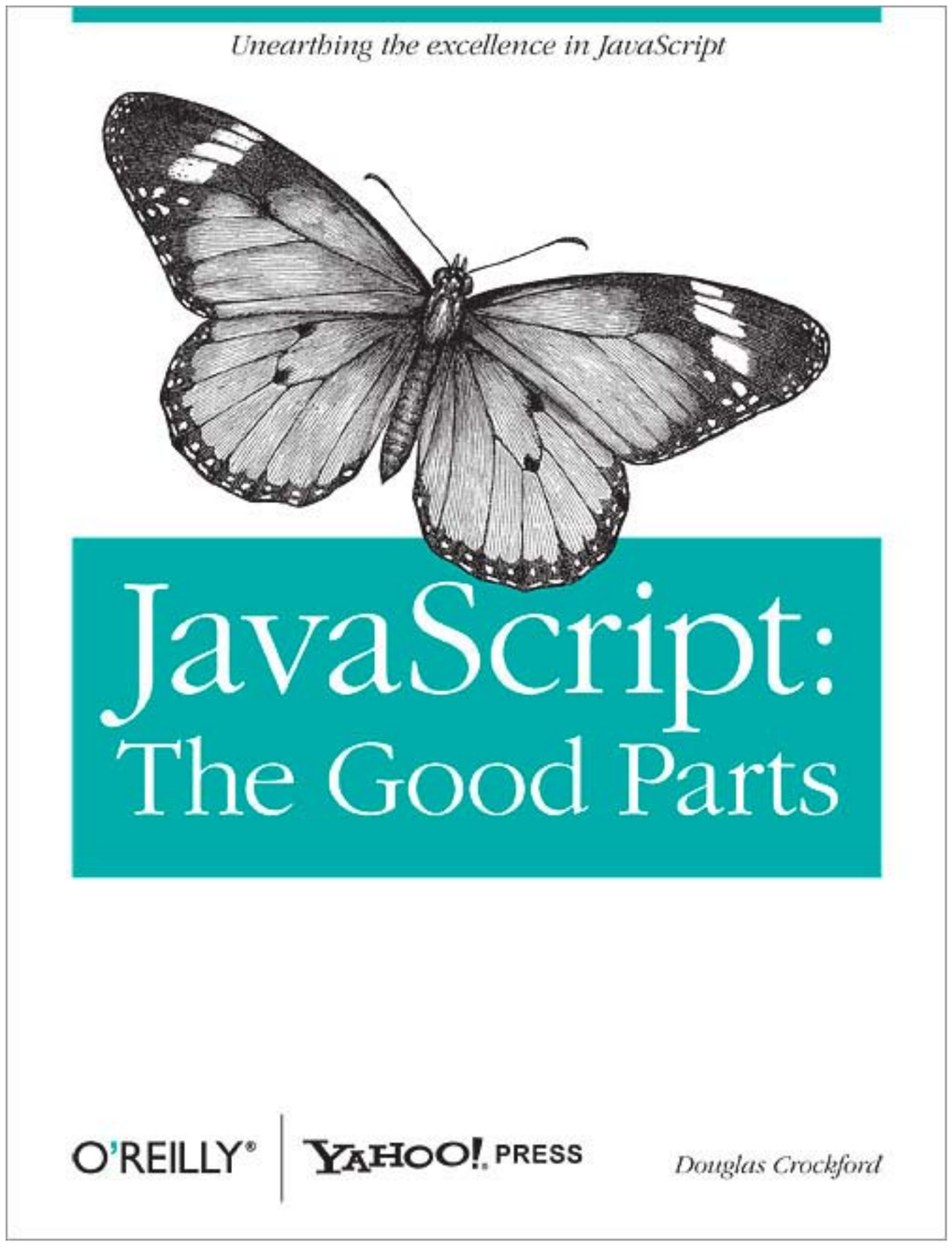
Javascript gives us ways to expand and manipulate it.

Eg) Namespaces

Single Global Variables

Object Literals

Nested namespaces



The Good Parts

eg) Closures

Passing information back and forth

Returning Functions

Reference instead of Create
via http://stackoverflow.com/questions/111102/how-do-javascript-closures-work

```
function foo(x) {
  var tmp = 3;
  return function (y) {
    alert(x + y + (++tmp));
  }
}
var bar = foo(2); // bar is now a closure.
bar(10);
```

The Bad Part

Gets Messy Quickly

Why?

SCOPE and @#%*(@#$* Global Variables

Not always consistent

eg) Semicolon insertion

```
// Good
return {
        javascript : "fantastic"
};
```

```
// Bad!
return
{
        javascript : "fantastic"
};
```

The Bad Part

Really easy to let it get away from you.

```
$(document).ready( function(){
    $('.success').hide();
    $('#bio').hide();
    if( (document.URL).search("[login]{5}") > 2) {
        $('.clockText').hide();
    } else {
        $('#login').hide();
    }
    //Request account
    $('#newRequest').click( function(e){
            e.preventDefault();
            request();
    });

    $('#bioExpand').click( function(e) {
        if(!$(this).hasClass('visible')) {
            $(this).fadeOut(100);
            $('#bio').slideDown(600, function() {
                $(this).find('p').delay(300).fadeIn(900);
            });

    }

});

$('input:not(#password)').keypress(function(e) {
    if(e.keyCode == 13) $(this).next().focus();
});

$('input#password').keypress(function(e) {
    if(e.keyCode == 13) request();
});

$('a.login').click(function(e) {
    e.preventDefault();
    $('.clockText').fadeOut(500, function() {
        $('#login').fadeIn(500);
    });
```

Structures and Exploration

http://forum.woodenboat.com/showthread.php?112363-Voronoi-Diagrams-in-Nature

You want to make a web app?

Lets forget about the code for a minute.

What are you trying to solve?

Who is going to use your application?

Where does your tool live?

How much time do you have to make it?

What does it actually do?

Sketch and Sketch and Sketch

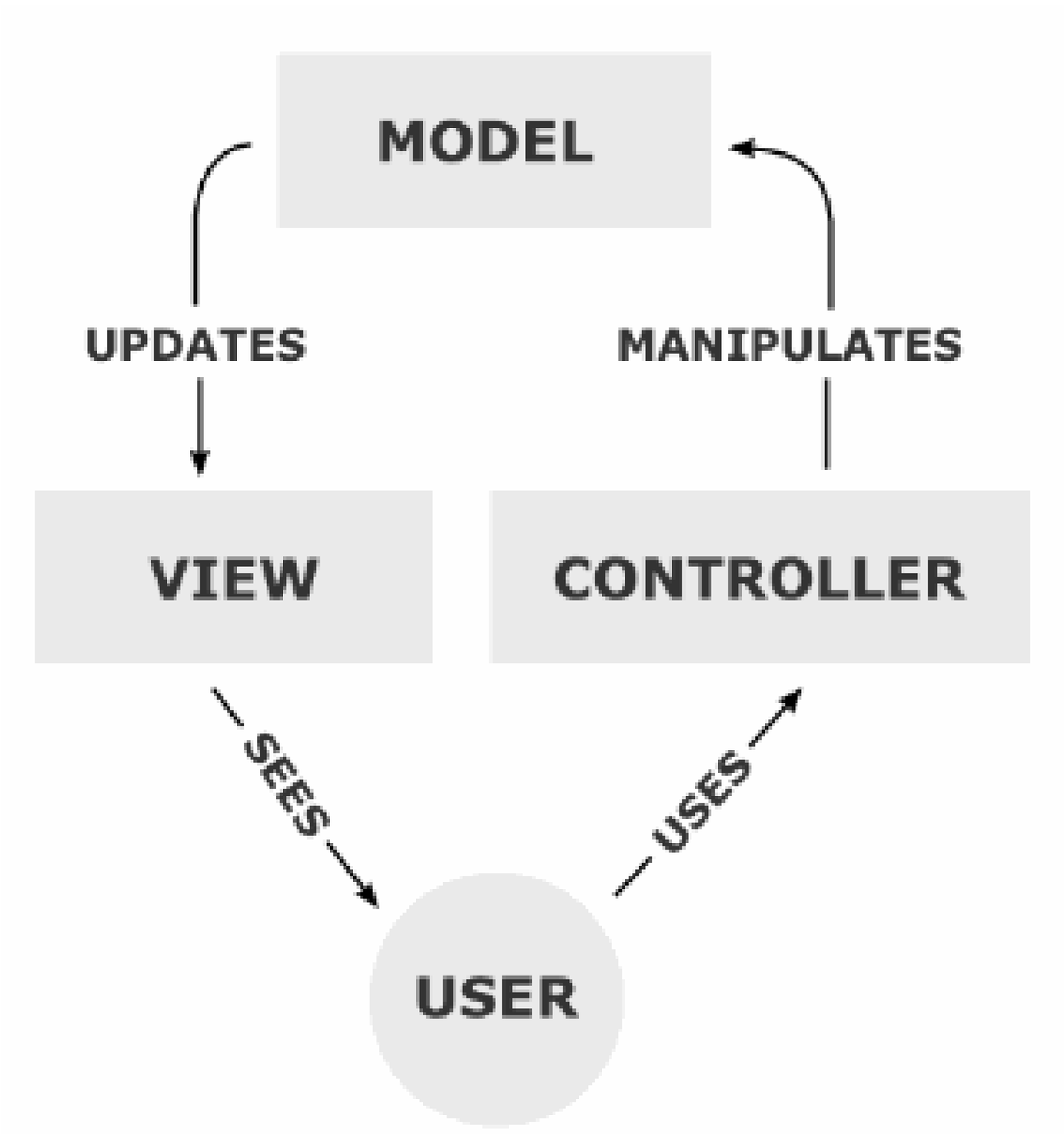
Formalize your sketching

List out what you need to do

Sketch a bit more

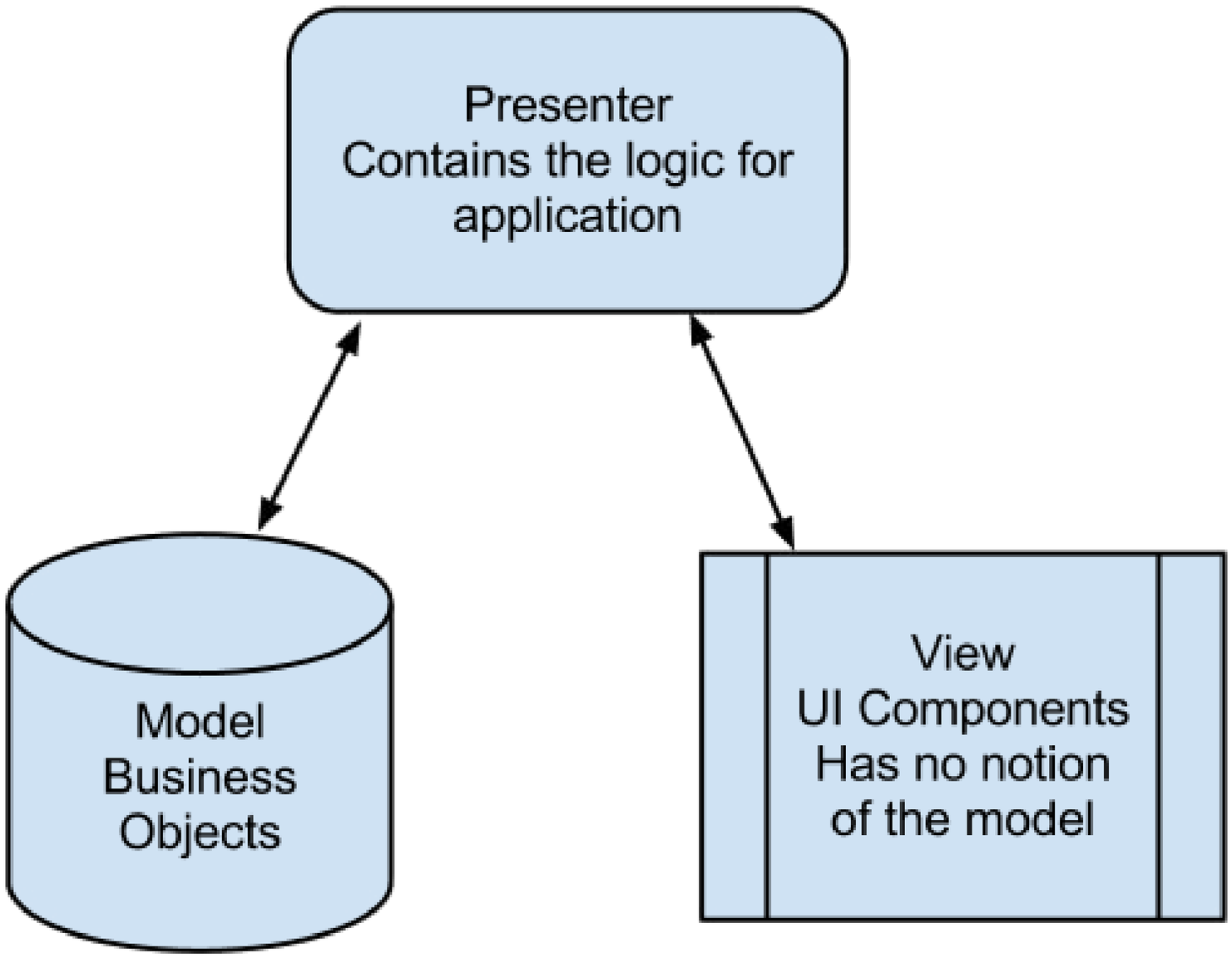
Let’s build something now...?

Model View Controller



http://www.codinghorror.com/blog/2008/05/understanding-model-view-controller.html

Model View Presenter



Model View *********@

BackboneJS

Intro to Backbone

What is Backbone?

Backbone is an awesome frameworks for organizing javascript projects.

Set of tools that help you manage complexity.

Set of methods and formalized conventions that are a given in modern Javascript

Powerful way to approach building Single Page Applications

Backbone Building Blocks

Model

Collection

View

Router

Backbone Model

Represents your data and what it's supposed to do.

The idea of the thing, not the thing.

Holds defaults, holds checking functions.

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

    defaults: {
        title: '',
        completed: false
    },

    toggle: function() {
        this.save({
            completed: !this.get('completed')
        });
    }

});
```

Backbone Collection

Collections hold groups of model data.

Says how these models are stored and where they go when they die.

Orders the data and lets you get information about it.

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

    model: app.Todo,

    localStorage: new Store('todos-backbone'),

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

    remaining: function() {
        return this.without.apply( this, this.completed() );
    },

    nextOrder: function() {
        if ( !this.length ) {
            return 1;
        }
        return this.last().get('order') + 1;
    },

    comparator: function( todo ) {
        return todo.get('order');
    }
});
```

Backbone View

Views are how the data and different parts of your application are represented.

How do you render out your ToDo, or your portfolio piece?

What are the different parts of your app? (This is why we sketch, by the way!)

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

    //... is a list tag.
    tagName:  'li',

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

    events: {
        'click .toggle':    'togglecompleted',
        'dblclick label':   'edit',
        'click .destroy':   'clear',
        'keypress .edit':   'updateOnEnter',
        'blur .edit':       'close'
    },

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

    render: function() {
        this.$el.html( this.template( this.model.toJSON() ) );
        this.$el.toggleClass( 'completed', this.model.get('completed') );

        this.toggleVisible();
        this.input = this.$('.edit');
        return this;
    },

    toggleVisible : function () {
        this.$el.toggleClass( 'hidden',  this.isHidden());
    },

    isHidden : function () {
        var isCompleted = this.model.get('completed');
```

Backbone Router

Routers help you manage states.

States are how your application know what's what.

Our different states:
Standing
Sitting
Lying down
Sleeping
Dreaming

How are those states represented?

```
var Workspace = Backbone.Router.extend({
    routes:{
        '*filter': 'setFilter'
    },

    setFilter: function( param ) {

        app.TodoFilter = param.trim() || '';


        app.Todos.trigger('filter');
    }
});

app.TodoRouter = new Workspace();
Backbone.history.start();
```

And ‘lo, magic.

Framework to form

Challenge is to translate the framework that Backbone gives us into the final application.

Lets look at:

- Data
- Templates
- Behaviour

Data and your app

Most modern applications require a database.

Localstorage is an HTML5 spec that lets you keep and persist info all within the client side, i.e. the browser. But doesn't let you share across.

Backbone lets you drop in localstorage first, and worry about the DB later.

eg) Meaning Clock started off as a localstorage app before I wrote the NodeJS server with a MongoDB database.

This is why we make lists. They will help you decide what is best for you.

Backbone.sync

Backbone.sync is a method to synchronize your collections and models across systems.

So, how do I make this work with my server and data?

```
var methodMap = {
      'create': 'POST'
      , 'update': 'PUT'
      , 'delete': 'DELETE'
      , 'read': 'GET'
};
```

Comes with a default way to manage data via the CRUD method.
Great for Node or Ruby on Rails.

Can be overridden if you want something custom.

Local vs. Remote storage

```
app.get('/clock/api', ensureAuthenticated, function(req, res) {
    console.log('Looking for ' + req.user._id)
    return Entry.find({ 'userid': req.user._id }, function(err,doc) {
        console.log('Found stuff!' + doc );
        if(!err){
            return res.send(doc);
        } else {
            return console.log(err)
        }
    })
});

app.post('/clock/api', ensureAuthenticated, function(req, res) {
    var newEntry = new Entry();
    newEntry.userid = req.user._id;
    newEntry.duration = req.body.duration;
    newEntry.meaning = req.body.meaning;
    newEntry.date = new Date(req.body.date);

    newEntry.save( function(err) {
        if(err) console.log("Error saving: " + err)
        res.json('Saved');
            });
});

// Modify meaning
app.put('/clock/api/:id',ensureAuthenticated, function(req,res) {
    return Entry.findById( req.params.id, function(err,doc) {
        doc.meaning = req.body.meaning;
        doc.duration = req.body.duration;
        return doc.save(function(err) {
            if(!err) {
                console.log('Updated successfully: ' + req.params.id )
            } else {
                return console.log(err)
            }
        })
    })
```

```
<script type="text/javascript" src="backbone.js"></script>
<script type="text/javascript" src="backbone.localStorage.js"
```

Create your collections like so:

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

  localStorage: new Backbone.LocalStorage("SomeCollection"),

  // ... everything else is normal.

});
```

Templates

Templates are how you separate HTML from JS.
Make sense of the different KNOWN (again, sketching) states of your app.

MyApp.js

```
    /**********************
    * View to a model *
    **********************/
var MeaningView = Backbone.View.extend({
    tagName: "li"
    , className: 'meaning-item'
    , template: _.template($('#meaning-item').html())
    , blankTemplate: _.template($('#blankMeaning').html())
```

Index.jade

```
<!-- templates -->
script#meaning-list(type='text/template')

script#dayView(type='text/template')
    ul.stats
        li.totalTime
        li.percentMeaningful
        li.keyWords

script#meaning-item(type='text/template')
    a.btn.hide.delete.onHover.pull-right x
    p
        <%= meaning %> for <%= duration %> hours
    .date.hide <%= date %>
    input.edit(type="text", placeholder="meaning").hide

script#blankMeaning(type='text/template')
    div
        a.btn.hide.delete.onHover.pull-right add Meaning
        h1 Have you done anything meaningful today?
```

Underscore.js

Framework for dealing with data.

Powers a lot of the BackboneJS tools, and is hugely worth learning.

Strong way to work with models and views.

That el **thing**

The el element refers to a DOM element in the backbone.view

Few different ways to deal with it.

$el is an easy to use jquery object

this.el is the standard Javascript DOM object.

I often do

this.$el.someJqueryFunction().anotherChainedjQueryFunction();

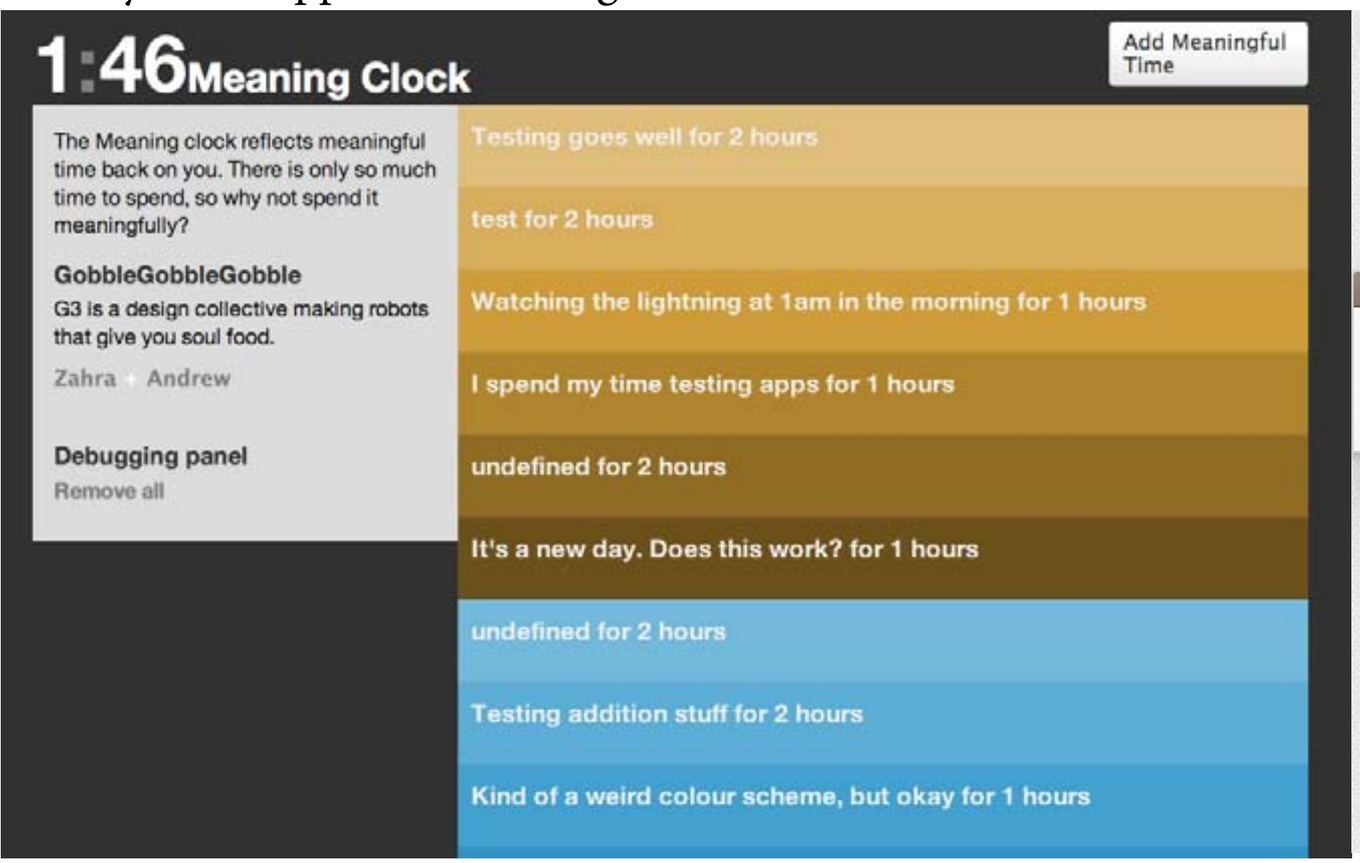
another option is this

```
$(this.el).html(this.template);
```

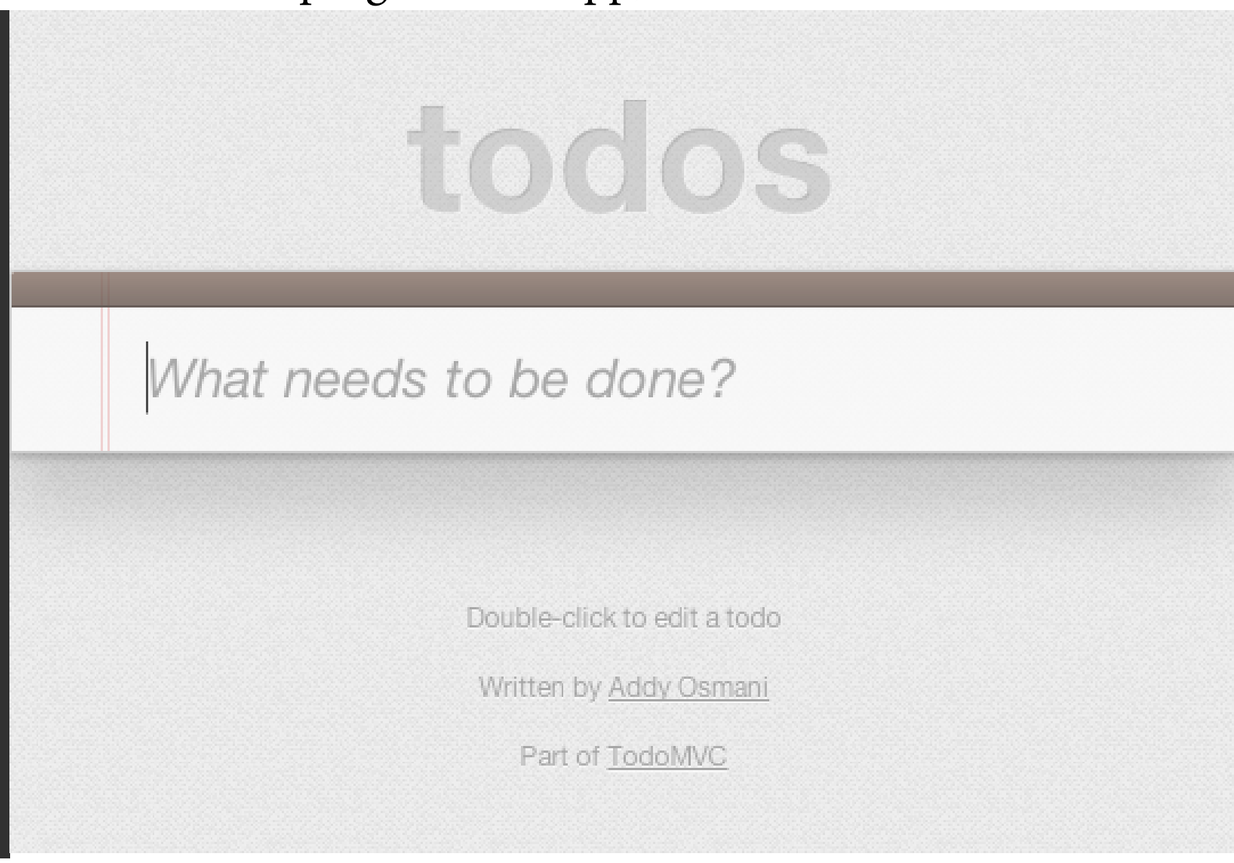
This reners the template defined in
el: $('#inviteToEnter')
, template: _.template($("#blankMeaning").html())
to #inviteToEnter

Lets Look At Some Code

My Own App, The Meaning Clock



A much better programmed app, todoMVC



Things to remember

Backbone is a FRAMEWORK, not a solution per se.
It helps you organize your thinking and avoid getting overwhelmed if you use it right.

Think and Sketch first, code later.
It's hugely valuable to do both, but drawing your app to scale on a piece of paper and pretending to interact with it is a hell of a lot more time effective than coding the wrong thing eight times.

Use documentation and resources
See the github repo for resources,
http://github.com/readywater/backbonejs_workshop

Andrew Lovett-Barron

@readywater

http://readywater.ca

http://relaystudio.com

http://github.com/readywater