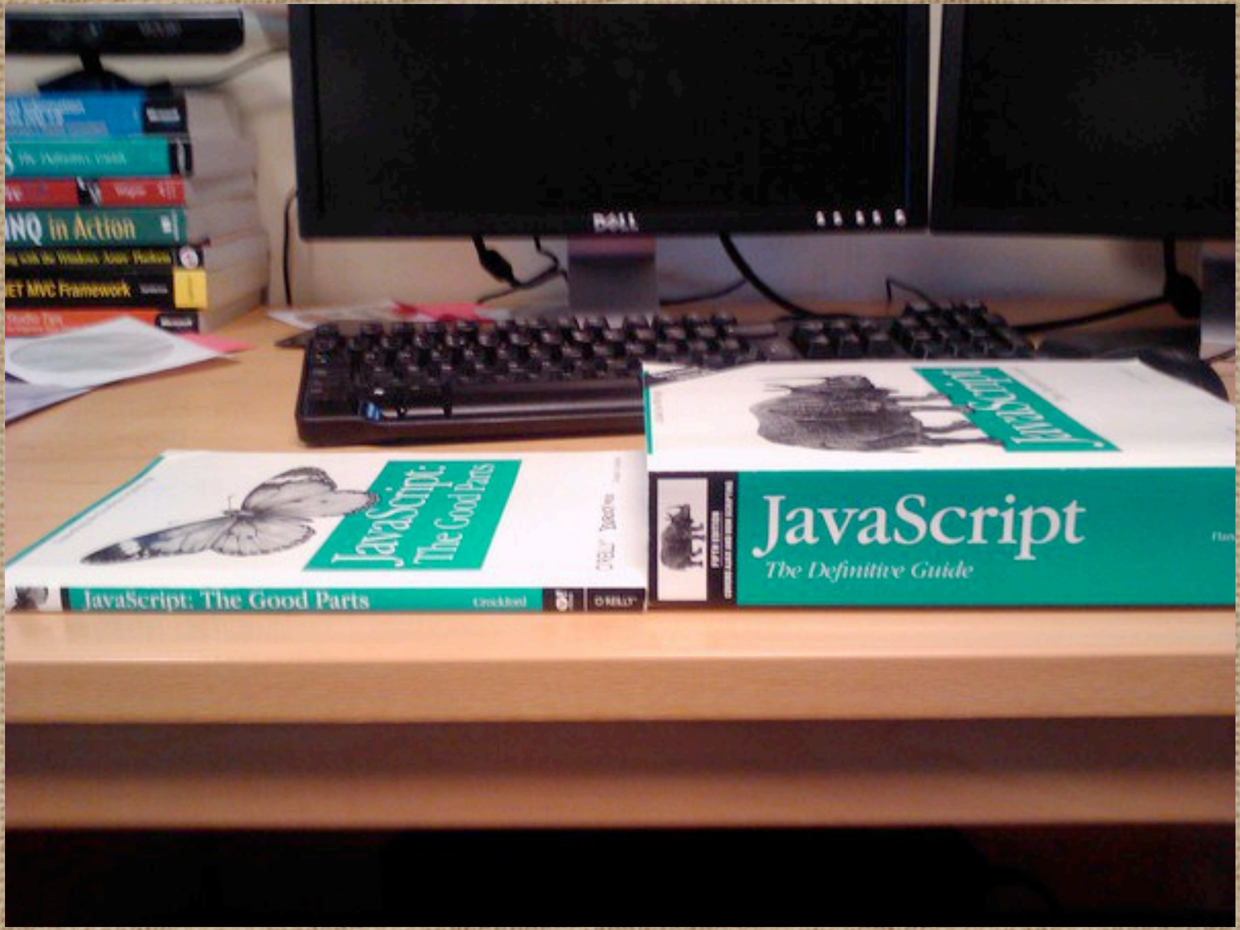
Decent Javascript

With jQuery, QUnit, and backbone.js

Mike Verdone @sixohsix
October 2011

example code:
https://github.com/sixohsix/backbone-example



"Fact:" Half of <u>Javascript: The Good Parts</u> is an appendix called The Awful Parts

"Fact:" half of the rest of the book is now considered bad advice

Good use of libraries can hide the suck of JS

- jQuery - client-side JS Swiss-army knife
- backbone.js - client-side MVC
- QUnit - dead simple unit testing

backbone.js

- client-side MVC is different than the web app MVC we know

TG2-style MVC

View
rendered HTML

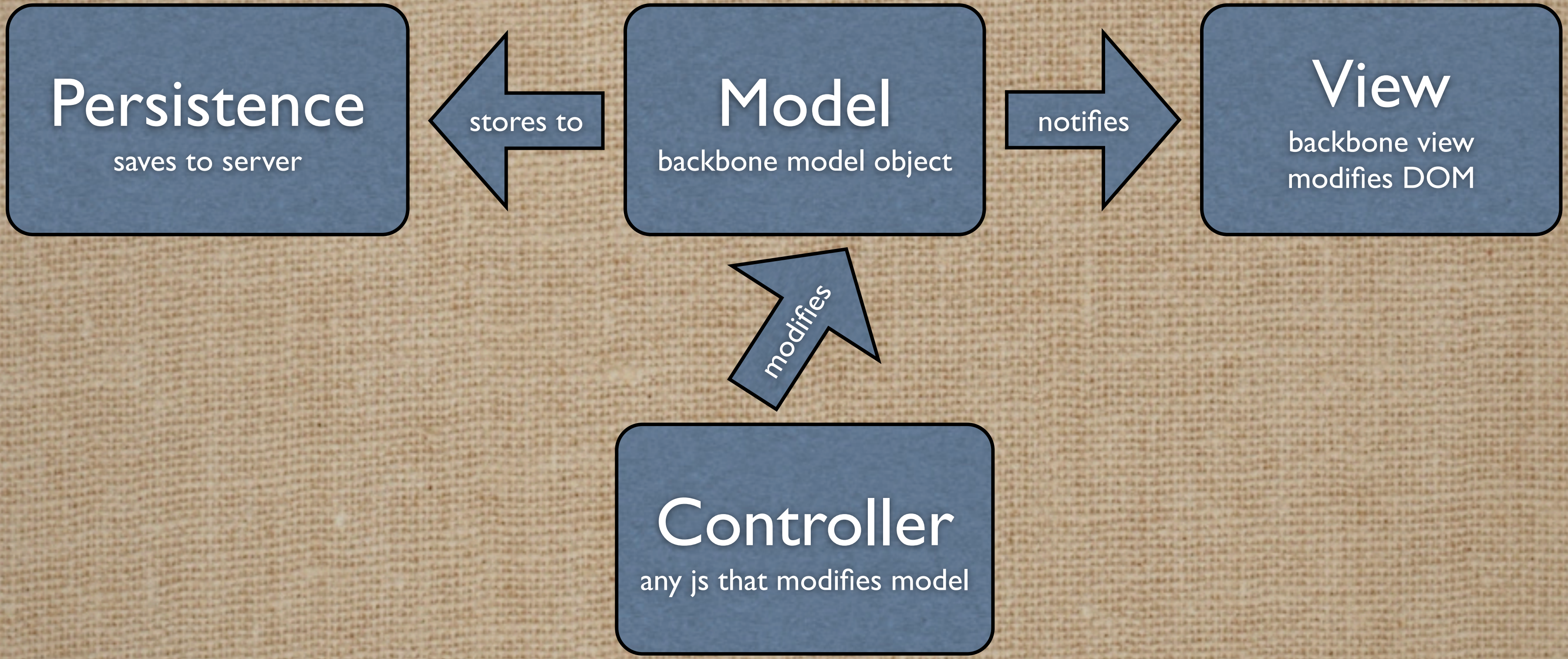
Model
server-side DB

Controller
CGI interpreter

TG2-style MVC

- monolithic: controller action redraws entire page
- most work done on the server side
- controller action drives the view

backbone MVC



backbone MVC

- all on the client side
- many Models, Views, Controllers on a single page
- uses the Observer pattern and events to update Views

Example Project

- an incrementing counter
- yes, it's boring. Deal with it.

First: this

- ensure you have a sane editor
- it *must* ensure your code is up to the *jslint* spec
- emacs + mooz's js2-mode = ♥

https://github.com/mooz/js2-mode

Note: keep your scope clean

```
/* All JS files should have this jQuery container to prevent global
 * scope leakage.
 */
(function($) {

   // your code here

})(jQuery);
```

The Model

```javascript
var CounterModel = Backbone.Model.extend({

    // Include sensible defaults for your model.
    defaults: {
        count: 0
    },

    // Define any custom model functions here.
    increment: function () {
        // Whenever we change the model we MUST use .get and .set
        // methods. Altering model.count does NOTHING.
        this.set({
            count: this.get("count") + 1
        });
    }
});
```

The Model

- Stores state
- Read values with .get
- Set values with .set
- When values are set, it emits a "*change*" event

The View

```
var CounterView = Backbone.View.extend({
    initialize: function (attributes, options) {
        // This super call sets this.model and this.el.
        Backbone.View.prototype.initialize.call(this, attributes, options);

        // This is what almost always happens in View
        // initialize. We bind our `render` to the model's change
        // event. Whenever the model changes, render is called.
        this.model.bind("change", this.render, this);
        this.render();
    },
    render: function () {
        $(this.el).html("Count: " + this.model.get("count"));
    }
});
```

The View

- Observes a model
- Has a DOM element called “*el*”
- When the observed *model* emits a *change event*, the view modifies *el* to present the new state

The Controller

```
<button onclick="$('#the_counter').data('counter_model').increment()">
  Click me to increment
</button>
```

The Controller

- Any JavaScript function that modifies the model is a controller

jQuery Plugitization

```
$.fn.setup_counter = function () {
    this.each(function (idx, el) {
        var model = new CounterModel();
        var view = new CounterView({ model: model, el: el });

        // Make the model and view available to the outside world
        // using the $.data functions.
        $(el).data("counter_model", model);
        $(el).data("counter_view", view);
    });
};
```

jQuery Plugitization

- Don't clutter the global scope!
- Hide Model and View construction behind a jQuery plugin
- Get Model and View objects later from the DOM node

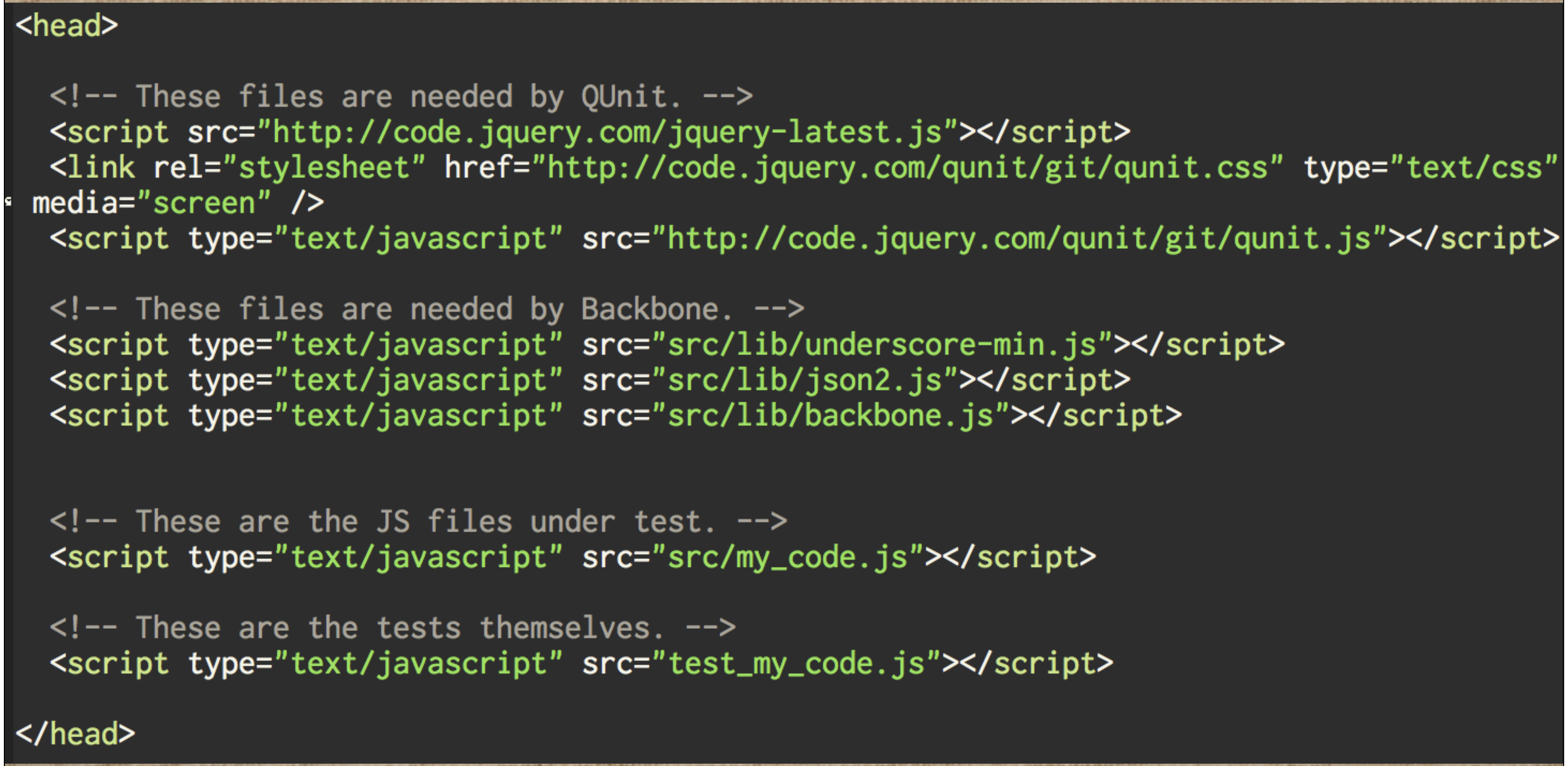
YES GREAT
NOW RUN IT IN
BROWSER

no,
write tests first

QUnit: setting it up

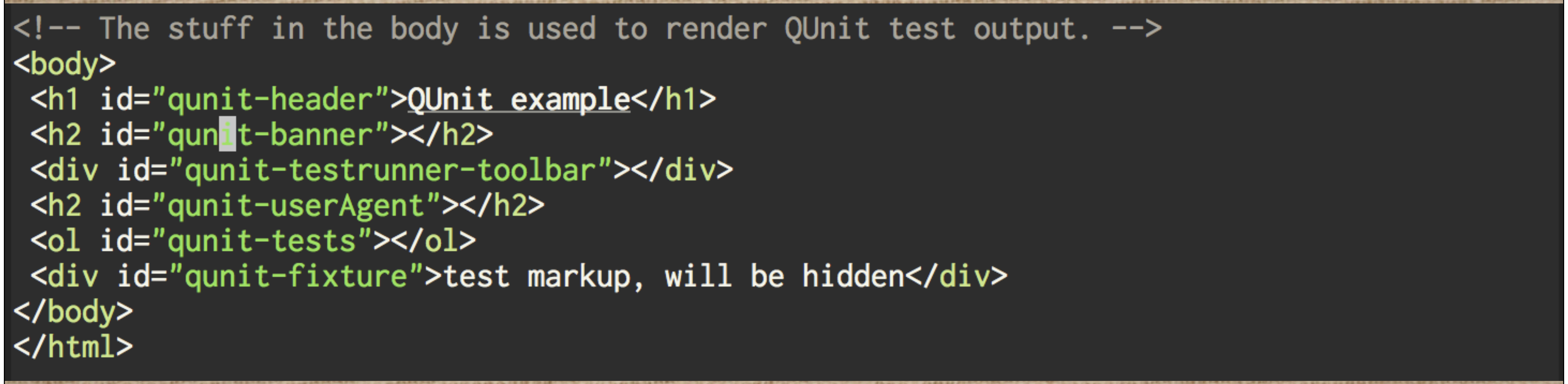
- We create an HTML file
- When you open this file in your browser, the tests are run
- Pretty sweet

QUnit: setting it up



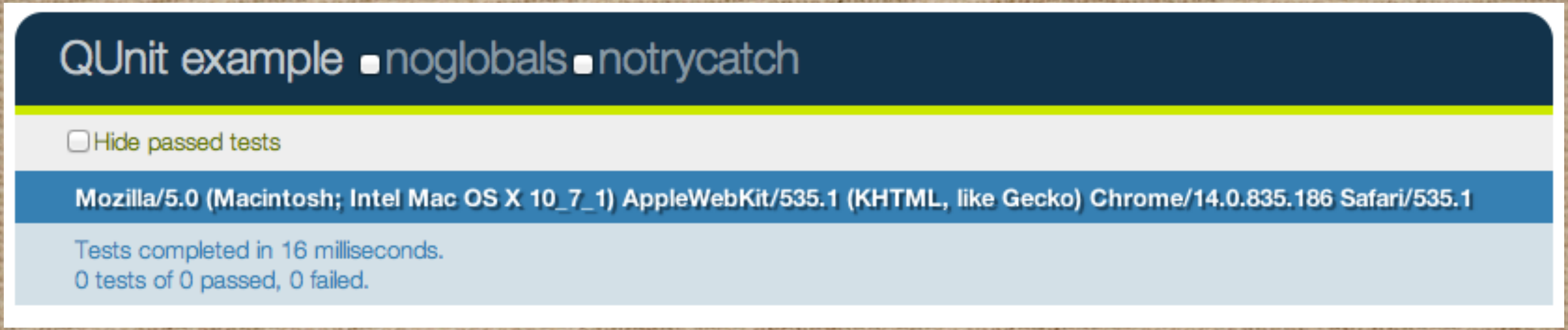
```
<head>

  <!-- These files are needed by QUnit. -->
  <script src="http://code.jquery.com/jquery-latest.js"></script>
  <link rel="stylesheet" href="http://code.jquery.com/qunit/git/qunit.css" type="text/css"
 media="screen" />
  <script type="text/javascript" src="http://code.jquery.com/qunit/git/qunit.js"></script>

  <!-- These files are needed by Backbone. -->
  <script type="text/javascript" src="src/lib/underscore-min.js"></script>
  <script type="text/javascript" src="src/lib/json2.js"></script>
  <script type="text/javascript" src="src/lib/backbone.js"></script>


  <!-- These are the JS files under test. -->
  <script type="text/javascript" src="src/my_code.js"></script>

  <!-- These are the tests themselves. -->
  <script type="text/javascript" src="test_my_code.js"></script>

</head>
```

QUnit: setting it up



```html
<!-- The stuff in the body is used to render QUnit test output. -->
<body>
 <h1 id="qunit-header">QUnit example</h1>
 <h2 id="qunit-banner"></h2>
 <div id="qunit-testrunner-toolbar"></div>
 <h2 id="qunit-userAgent"></h2>
 <ol id="qunit-tests"></ol>
 <div id="qunit-fixture">test markup, will be hidden</div>
</body>
</html>
```

Open it in your browser...



QUnit example ☐ noglobals ☐ notrycatch

☐ Hide passed tests

Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_1) AppleWebKit/535.1 (KHTML, like Gecko) Chrome/14.0.835.186 Safari/535.1

Tests completed in 16 milliseconds.
0 tests of 0 passed, 0 failed.

QUnit Tests

```
(function($) {

    module("Counter tests");

    test("can set up a counter with jQuery plugin", function () {
        var el = $("<div />");
        $(el).setup_counter();

        var model = $(el).data("counter_model");
        var view = $(el).data("counter_view");

        // The `ok` function verifies that an expression is true.
        ok(model !== undefined, "The model is accessible");
        ok(view !== undefined, "The view is accessible");
    });
```

QUnit Test Functions

- Four functions to learn:
  - module(n) - Group all following tests into a module named *n*
  - test(n, f) - Declare a test named *n*, executed by calling *f*

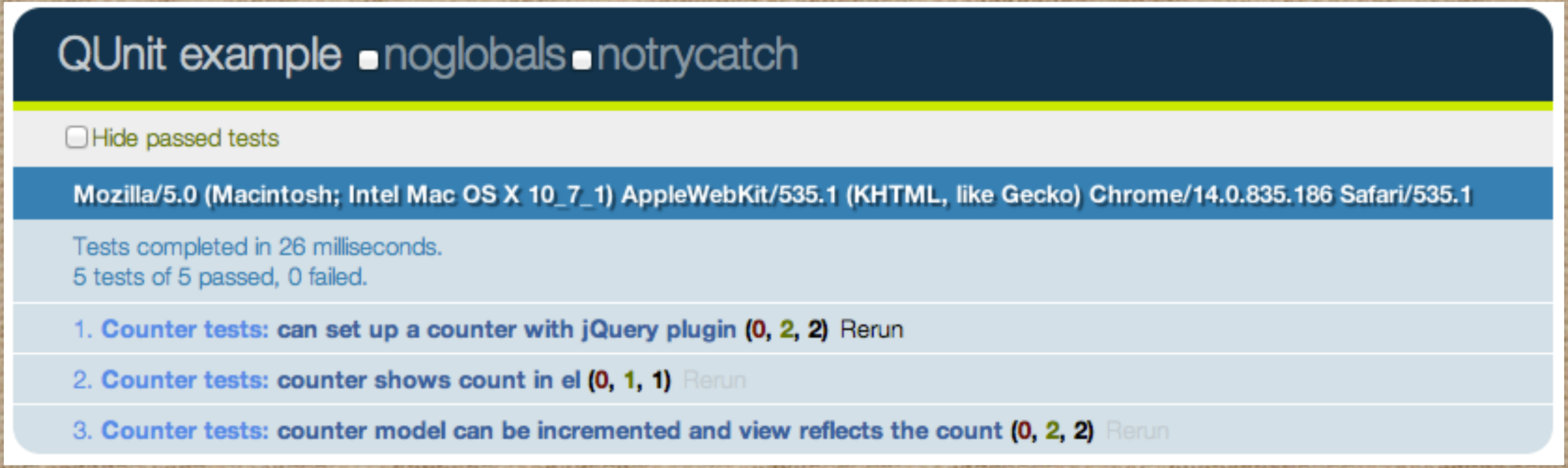
QUnit Test Functions (2)

- ok(expr, msg) - Test that *expr* is true, show *msg* if false
- equal(a, b, msg) - Test that *a* equals *b*, show *msg* if false

QUnit Tests (2)

```
test("counter model can be incremented and view reflects the count", function () {
    var el = $("<div />");
    $(el).setup_counter();

    var model = $(el).data("counter_model");
    model.increment();

    equal(1, model.get("count"), "The model's count is one");
    equal("Count: 1", $(el).html(), "The counter el shows a one count");
});
```

Run it in your browser again...



QUnit example ■noglobals■notrycatch

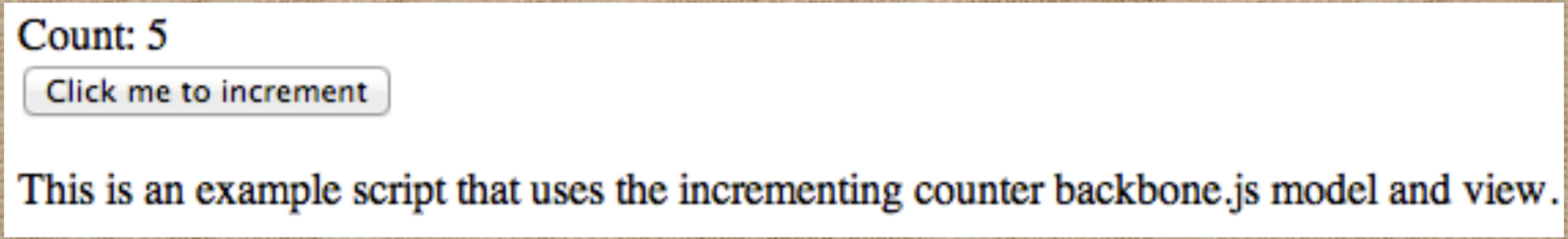
☐Hide passed tests

Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_1) AppleWebKit/535.1 (KHTML, like Gecko) Chrome/14.0.835.186 Safari/535.1

Tests completed in 26 milliseconds.
5 tests of 5 passed, 0 failed.

1. Counter tests: can set up a counter with jQuery plugin (0, 2, 2) Rerun
2. Counter tests: counter shows count in el (0, 1, 1) Rerun
3. Counter tests: counter model can be incremented and view reflects the count (0, 2, 2) Rerun

Incidentally, it works...


Count: 5

This is an example script that uses the incrementing counter backbone.js model and view.

More features

- collections: ordered sets of models
- serialization to/from a server
- URL manipulation/HTML5 History

Keeping our JS Sane

- Write tests. If your code is untestable, it probably smells.
- Use backbone.js to keep your code modular and testable.
- Use jQuery plugins to keep the global scope clean.