Simple JavaScript abstractions

I'm Kim

I work at BEKK

I'm on Twitter: @kimjoar

I'm on Github: @kjbekkelund

Stop using jQuery directly all over your code!

Why?

```
jQuery(function() {
  $(“.user form”).submit(function(e) {
    e.preventDefault();
    $.ajax({
      // ...
      success: function(data) {
        var name = data.name ? data.name : “Unknown”;
        $(“.user .info”).append(“<h1>” + name + “</h1>”);
      }
    });
  });
});
```

Why?

PAGE EVENT

USER EVENT

NETWORK I/O

NETWORK EVENT

PARSING RESPONSE

TEMPLATING

```
jQuery(function() {
  $(".user form").submit(function(e) {
    e.preventDefault();
    $.ajax({
      //
      success: function(data) {
        var name = data.name ? data.name : "Unknown";
        $(".user .info").append("<h1>" + name + "</h1>");
      }
    });
  });
});
```

This is a mess

“Internal quality is what lets us cope with continual and unanticipated change”

Steve Freeman, Nat Pryce

Growing Object-Oriented Software, Guided by Tests

We need decoupling

We need structure

We need abstractions

Abstracting views

A view's responsibility:

It owns an HTML element

Views can read from the DOM, change it, and listen for events

Our view API

```
var el = $(“.user”);
var userView = new UserView(el);
userView.showUserInfo();
```

UserView is now responsible for
everything related to the DOM **within** `.user`

View API implementation

```
var UserView = function(el) {
  this.el = el;
}

UserView.prototype.showUserInfo = function() {
  this.el.append("<h1>Kim Joar</h1>");
}
```

That's right — it's just convention

... but how will the view get any data?

Events!

Events

```
var events = new EventEmitter();
var UserView = function(el) {
  this.el = el;

  // Bind events
  events.addListener(“user:fetched”, this.showUserInfo, this);
}

UserView.prototype.showUserInfo = function(user) {}

// Trigger events
events.emit(“user:fetched”, { name: “Kim Joar” });
```

An example

```
jQuery(function() {
  new UserView($(“.user”));
  events.addListener(“user:submit”, getUser());
});
function getUser() {
  $.ajax({
    // ...
    success: function(data) {
      events.emit(“user:fetched”, data);
    }
  });
}
```

Or, even better

```
jQuery(function() {
  var user = new User(); // user model!
  new UserView($(“.user”), user); // view is allowed to
});                               // call methods on model
```

Or, even better

```
jQuery(function() {
  var user = new User();
  new UserView($(“.user”), user, ...);
});
```

And similarly we can also pass in
other dependencies as arguments
(events are not always the answer)

Our views

≈

Presenter or Controller

We have encapsulated network requests in **Models**.

We have encapsulated the DOM in **Views**.

We communicate using **events** instead of relying on callbacks.

We have code that is far easier to **test**.

And we are on our way to an **MV*** architecture.

Let’s add some sugar

Localized lookup

```
// as views are passed `el`, which is a jQuery object,
// we can search through the descendants as follows:
userView.el.find(“h1”)

// … but if we add:
UserView.prototype.$ = function(selector) {
  return this.el.find(selector);
}

// we can do this instead:
userView.$(”h1”) // less coupled to the DOM!
```

Event delegation

```
UserView.prototype.events = {
  “submit .user-form”: “validateAndPost”,
  “click .title”: “editName”
};
UserView.prototype.validateAndPost = function() {}
// or (using Backbone.js):
var UserView = View.extend({
  events: {
    “submit .user-form”: “validateAndPost”
  },
  validateAndPost: function() {}
});
```

Local events

```
var UserView = function(el, user) {
  this.el = el;
  // Binding events on `user` instead of `events`
  user.addListener(“fetched”, this.showUserInfo, this);
}
```

```
User.prototype.addListener = function() {
  if (!this.events) this.events = new EventEmitter();
  this.events.apply(this.events, arguments);
} // and the same for `emit`

user.emit(“fetch”); // triggers the “local” event
```

Add sugar using layers,
mixins, or an existing library

If you like this, check out

"An element's **cohesion** is a measure of whether its responsibilities form a **meaningful unit**"

Steve Freeman, Nat Pryce

Growing Object-Oriented Software, Guided by Tests

Don’t use it less, use it smarter!

Questions?