responsive, real-time

with Backbone.js and WebSockets

Andrei Cîmpean, @Andrei_Cimpean

Today, we are going to look at how we are building a real-time, responsive JavaScript client app.
For Google Chrome.
And IE8.

What is a real-time application ?

- It enables users to receive information as soon as it is published

What is a responsive application ?

- Its UI updates instantly, no matter what

Why do we need *“real-time, responsive”*

- a client imports entries, packages them and puts them up on the market for another users to bid on and, if they win the auction, to pay
- an app that behaves like a market
- allows clients to sell and bid
- time constrained user actions (seconds)

What we needed to do

- structure the market on the client app
- communication mechanism
- don't stress the server
- support only Chrome and IE 8+ :|
- make sure it works at least a full day without crashing or requiring a full page refresh

Technology decisions

- Backbone.js + plugins – for structure
- CometD – for communication

Structure

Backbone.js + plugins

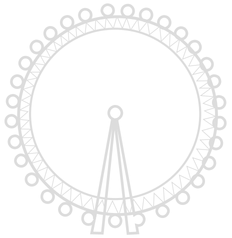
Backbone.js (http://backbonejs.org/)

BB is a JavaScript library with a RESTful JSON interface, and is based on the model–view–presenter (MVP) application design paradigm. It provides:

- models with key-value binding and custom events
- collections with a rich API of enumerable functions
- views with declarative event handling
- routing system with History API support

Backbone.js (http://backbonejs.org/)

```
// model definition
var Package = Backbone.Model.extend({
  doSomething: function() { ... }
});
// collection definition
var Repository = Backbone.Collection.extend({
  model: Package
});
```



Backbone.js (http://backbonejs.org/)

- Entry and Package as Models,
- Entries and Repository (fancy for packages) as Collections
- add an Entries collection in Package to satisfy relationship

Backbone.js Views

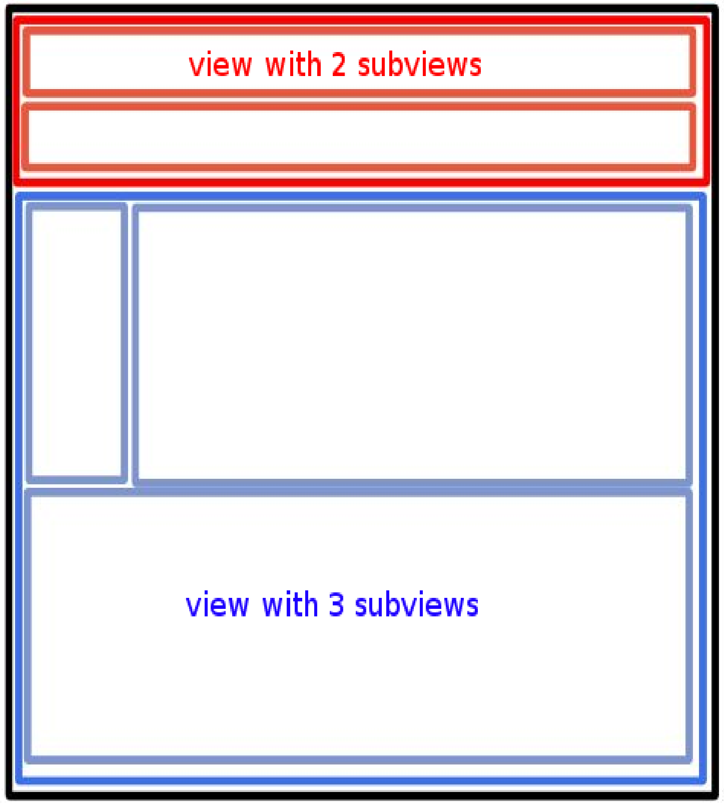
```
var PackageWithBid = Backbone.View.extend({
  events: {
    "click .button.accept": "accept",
    "click .button.reject":   "reject"
  },
  render: function () { ... }
  ...
});
new PackageWithBid({ model: repository.first() }).render().el; // DOM element
```

Backbone.js Views

- views can be binded to model or collection events
- managing a large number of nested views can be a pain

Backbone.js LayoutManager
(https://github.com/tbranyen/backbone.layoutmanager)

- a better way to structure views
- easy to set up
- render() is now managed
- gain beforeRender() and afterRender()
- manages cleanup



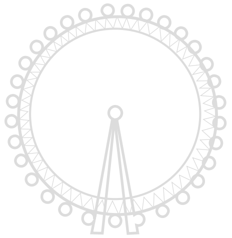
layout with 2 subviews

Nesting Views

```
var ViewBids = Backbone.View.extend({
  manage: true, // needed by layout manager for our approach
  beforeRender: function() {
    this.collection.each(function() {
      this.insertView(new PackageWithBid());
    }, this);
  },
  serialize: function() {
    return { package: this.model };
  }
});
```

Why Backbone.js ?

- Backbone.js was chosen because it is small, very light; The entire source code can be read with ease (~ 1500 LOC)
- good documentation
- a big community with a lot of plugins, some really good



Communication

WebSockets

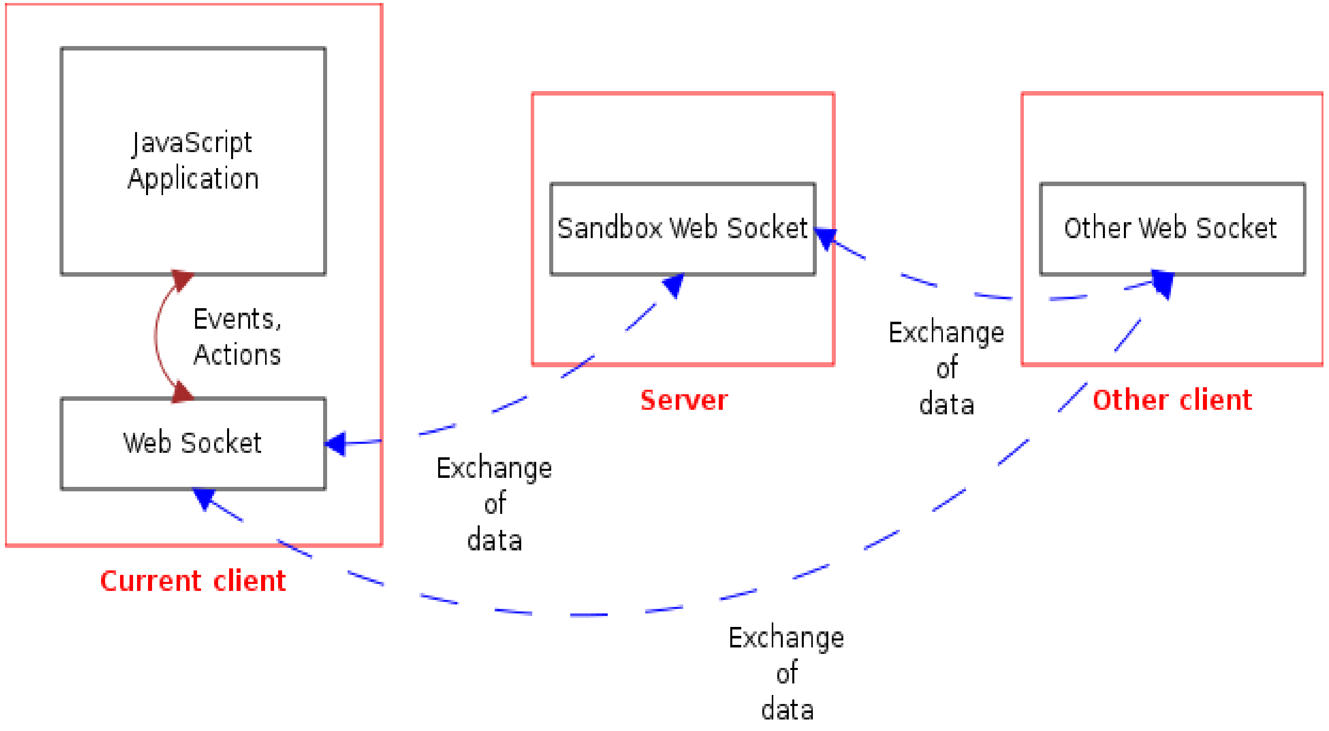
“It has to be a bit more complicated.”

Internet Explorer Dev Team

WebSockets (http://www.websocket.org/)

- a full-duplex single socket connection over which messages can be sent between client and server
- doesn't work in Internet Explorer

WebSockets



JavaScript Application
Events, Actions
Web Socket
Current client
Sandbox Web Socket
Server
Other Web Socket
Other client
Exchange of data
Exchange of data
Exchange of data

Sorting out IE8+

- any browser with Flash can support WebSocket using a web-socket-js shim/polyfill
  - flash leaks memory
- fallback system: long-polling vs flash
  - flash leaks memory
- use a “wrapper” that enables fallback

CometD (http://cometd.org/)

- scalable HTTP-based event routing bus that uses a Ajax Push technology pattern known as Comet
- provides meta-channels for error messages
- provides channels that can be used to filter content update by subscribing to them:

```
var sub = socket.subscribe('/foo/bar/', function() { ... });
cometd.unsubscribe(sub);
```

Connecting to a CometD server

```
function connect(path) {
  window.socket.configure({ url: path });

  window.socket.addListener('/meta/subscribe',metaSubscribe);
  window.socket.addListener('/meta/handshake', metaHandshake);
  window.socket.addListener('/meta/connect', metaConnect);

  return window.socket.handshake({
    ext: { authentication: { user: username, sessionid: sessionid } }
  });
}
```

Architecture decisions

- use comet channels
  - an update channel, where bids on the client's repository are sent
  - an offer channel, where new packages are sent for other clients to bid on

Architecture decisions

- pass-through REST-like server:
  - GET returns appropriate responses
  - POST, PUT, DELETE
    - 200 OK - if server is not dead
    - confirmation on a channel will be received as soon as possible

Optimism and responsiveness

- we are optimists
- each action that would require a confirmation is actually displayed as if it was a success immediately
- if the confirmation message says that something failed, we just display a notification and revert. **Most of the time this is not required.**

Reverting optimism

- when the action is performed, after the server call we replace old views with ones representing the new state (one property)
- if the confirmation invalidates the action, we restore the previous state and re-render all
- this approach guards almost all the actions a user can perform in the application

Reverting optimism

- actions that change more than just an order state, first create a snapshot of itself
- if the confirmation invalidates the action, we restore the snapshot and re-render all
- for these, we render the view representing the new state but block them with a loading animation

So, basically

- be optimistic when rendering but prepared to raise alarms
- always use ;