Model-View-Controller

WebSockets
ah, JavaScript

- historically, a simple place
- doing just fine without patterns and frameworks
- a victim of its own potential
- nothing is simple anymore
- patterns combat complexity
#singlepageappproblems
SPAs today

everywhere!

more SPA-like behavior in static pages

third-party tools

systems of dependencies

easy to set up, but then...?
win some/lose some

- time to first render
- SEO
- managing HTTP requests
- creating good packages
implementation issues

 creading app scaffolding
 ➡️ how & when to redraw
 ➡️ where client interactions fit
 ➡️ those darn HTTP requests
some approaches

- twitter
  - server-rendered HTML, Flight

- airbnb
  - Node, full-stack Backbone

- meteor (ok, not an implementation)
  - Node, MVC-style models, EDA-style messaging
two households, both alike in dignity..
Model-View-*

- circa late 1970s
- closely tied to OOP
  - SmallTalk
- implemented in popular frameworks
- traditionally backend
Model

- aka “object”
- closely mirrors data
- track their own relationships like relational DBs
View

- presentational part of the app
- draw the interface
- interpolate data
- (optionally) capture interactions
we’ll get to that in a sec
MV* in JavaScript

- MV* follows OOP
- mix in a dash of rapid development..
- fast-forward to now
  - numerous options
  - range of values for *
  - few popular choices outside MV*
not-so-
the new mutiny
Event-Driven Architectures

- less formal than MVC
- suit high-I/O systems
- simple and reactive
- seen more in tools than applications
events and objects

- less data persisted overall
- models less bound to data
- natural "model" is a state
events themselves

- user interaction with view
- server interaction with state
- messaging within application
- highly decoupled
EDA in JavaScript

-how we do it

-all JS triggered by client-side events

-event handling has evolved over time

-inline onclick →

-event delegation, pub/sub

-dearth of sophisticated EDA JS frameworks
a pair of star-crossed lovers?
dealing with data

- MVC: everything in models
- EDA: models maybe?
- MVC: lends itself to CRUD
- EDA: ad-hoc updates
the views

- MVC: one-to-one with data
- EDA: highly composite view reflecting state
- MVC: view responsible for interaction
- EDA: interaction first-class component
the action

- MVC: triad-specific
- EDA: first-class
- MVC: objects don’t influence controllers
- EDA: objects have event awareness
everybody loves MV*
the whole family

- MVC: tight triads, view bubbles up through controller
- MVP: model and view paired, presenter can “float”
- MVVM: model and view super tightly coupled, observerish
the appeal

- the age of OOP
- big teams, separation of concerns
- the age of REST
- easy mapping to server code
- good tools, obvs
how we do it (now)

- lightweight MVC
- controller in router, view, other?
- presenter abstractions
- automatic binding
framework choices

- no need for boilerplate
- too much for small apps?
- too little for big apps?
- SoC depends on NoC
shaking the yoke of inauspicious *s
a controller

- Spine.js in TodoMVC
- interface events
- bindings to app functions
- manage model properties
- rendering, init
a presenter (kinda)

- Backbone.js in TodoMVC
- interface events
- observer events
- rendering, init
a viewmodel

- Knockback.js in TodoMVC
- interface binding events
- magic interface events
- observer events
- controller handles render, init
no man is an island
someone to talk to

- given: server supplies first render
  - data
  - HTML
- server updates
- localstorage
when it’s easy

- CRUD
- REST
- highly automatable
when it’s tricky

- changing multiple models
- lots of quick state changes
- uploading offline data
hey, it’s WebSocket!
what is that tho

- its own protocol

- TCP, port 80, ws: and wss:

- server can push data

- replacement for e.g. long-polling

- example: Word²
not your mama’s comet

- bi-directional
- no request, no response
- cross-origin support
- dead simple API
available now

- FF 16
- Chrome 23
- IE 10
- Opera 12
- Safari 6

iffy on mobile (iOS yes, Android no)
uses

tickers

chat

collaborative editing

yes, games

every Node Knockout app ever
getting cozier with the server
EDAing the *s

- controllers/presenters/viewmodels full of events
- free events from triads
- subscribe at application level
- any old event system will do
event piping

- decouple user and server interaction
- provide place for validation
- allow multiple subscribers
EDA + WebSockets

- chained like interface events
- specific controllers can subscribe to global events
- updates can be cached and bundled
time
real talk
DIY WebSockets

- declare a socket with URL and protocol
- listen for open connection
  - send client messages
- listen for server messages
- close the connection
DIY WebSockets

```javascript
var msgPane = document.querySelector("#msgPane"),
    s = new WebSocket("ws://yoursite.com", ["soap"]);

// listen for connection
s.onopen = function(e) {
    s.send("open for business!");
};

// listen for server message
s.onmessage = function(e) {
    msgPane.innerHTML = e.data;
};
```
on the server

- can also DIY
- becomes handy to use a utility
- niceties like broadcasting
- options for almost any backend
socket.io example

```javascript
var msgPane = document.querySelector("#msgPane");
    s = io.connect("http://yoursite.com");

    // listen for connection
    s.on("connect", function() {
        s.emit("open for business!");
    });

    // listen for server message
    s.on("newmessage", function(data) {
        msgPane.innerHTML = data;
    });
```
connecting it

▷ add messaging anywhere in controller/presenter/etc.

▷ use it for syncing if rolling your own

▷ replace framework sync calls in implementation with socket events

▷ use WebSockets plugin for framework (e.g. backbone.io)
sockets for control

- take action based on server messages
- pipe events
- subscribe on models and views
- very similar to MVP
var Events = function() {
    // ...
    // simple pub/sub
};

var sock = io.connect("http://yoursite.com");

// listen for server message
sock.on("newobject", function(data) {
    Events.publish("objectadded", data);
});
```javascript
var that = this;

Events.on("objectadded", function(data) {
    that.add(data);
});
```
var that = this;

Events.on("objectadded", function(data) {
    that.container.append(that.render(data));
});
together at last

- a tool to use within your pattern
- use a little (ad-hoc)
- use a lot (syncing)
- easy as pie to add
thanks!