Backbone Marionette

Arquitectura no limpia y REST no CRUD… ^_^

¡Katayunos!

Nueva edición el sábado 8 de febrero

www.katayunos.com

¿Hypermedia?

- Más bien **lógica en servidor** y recursos virtuales
- PUT sobre /validacion/ devuelve la evolución de la estructura de la reserva en una fase
- “validación” no es una tabla de base de datos

Marionette

El de las rimas con…

Quién

- @ Derick Bailey
- http://lostechies.com/derickbailey/
- Backbone Syphon
- Backbone Model Binder

Por qué

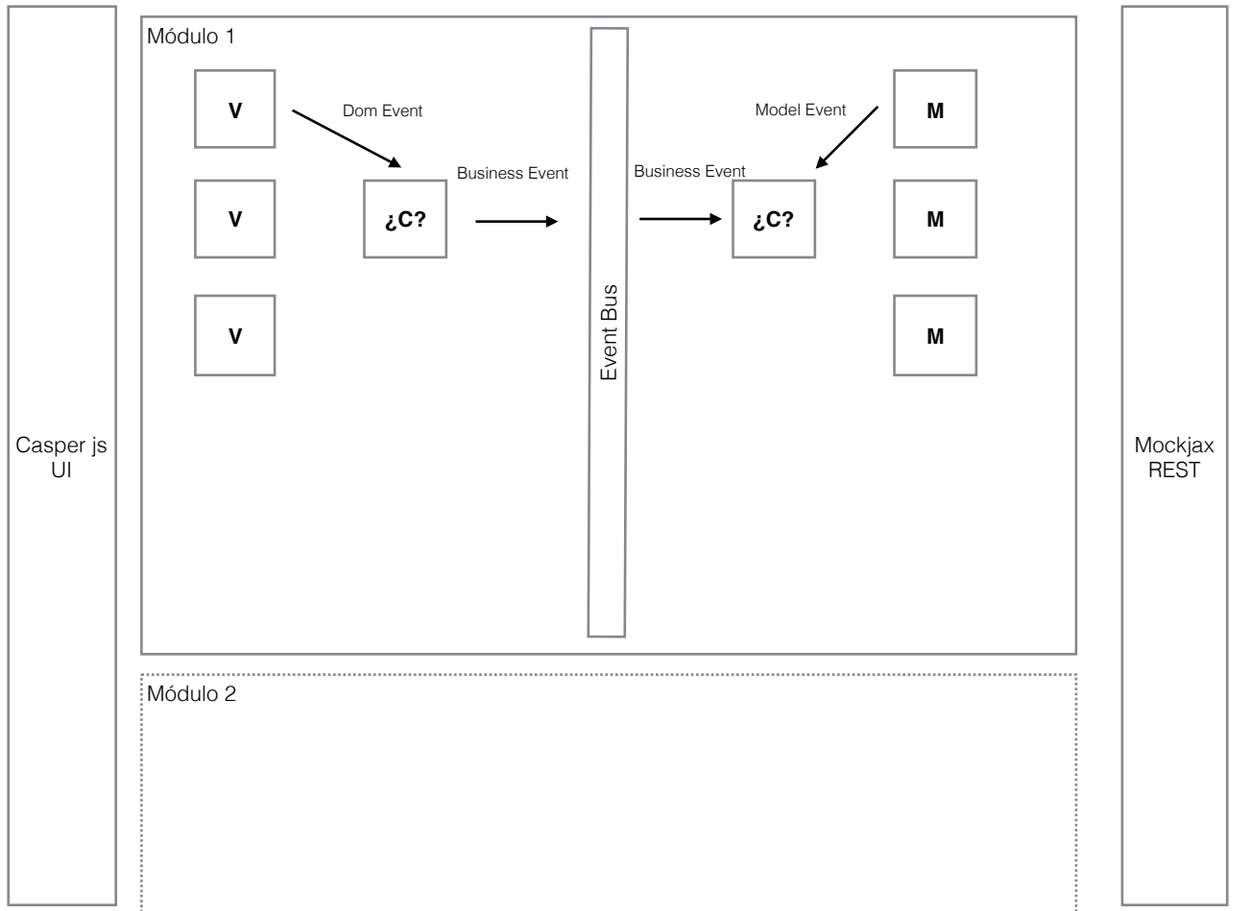
- Los problemas de backbone:
  - memory leaks, zombie events
  - vistas gordas
  - active record

Patrones usados

- event bus central
- command and query, events
- modules / modularity
  - **La clave**: los módulos suscriben a la app : eventos/comandos, inician views en regiones, etc.
- cached views
- CollectionView, ItemView, etc….

Active Record

- se mantiene :-(
- ideal para CRUDs
- no muy bien para testera
- ni para llamadas de lógica de negocio “no crud”


Casper js
UI
Módulo 1
V
V
V
Dom Event
¿C?
Business Event
Event Bus
Business Event
Model Event
¿C?
M
M
M
Módulo 2
Mockjax
REST

¿todavía backbone?

- mi consejo: quedarse más con los patrones que con el framework en si mismo. A nivel de diseño tiene algunas ideas muy potentes

Demo

Conclusiones

Hexagonal

- No porque queramos poder cambiar de framework
- Flujo:

    1. BDD del comportamiento desde el caso de uso con adaptadores falsos >> obtengo los puertos

    2. Tests funcionales contra servicios reales con adaptadores reales

Hexagonal

- Puertos & Adaptadores
  - El Puerto es una Interface como DataBase, Web o Device
  - El Puerto es el Single Entry Point a un “propósito” de la app
  - El Adaptador implementa el Puerto usando una tecnología concreta: ElasticDataBase, H2DataBase, jQueryWeb, SerialDevice

Una **Factoría Abstracta** puede servir como Single Entry Point para un puerto

Hexagonal

- Puertos
  - Tiene sentido organizar las posibles peticiones y respuestas alrededor de Puertos
  - Varias opciones para su implementación: API explícita o basada en eventos

Requests & Responses >> Puertos

Ejemplos en código:

- Request fichas.crear en ng/fichas/ListaController.js
- Response transitionWith en app/FichasApp.js

Se podrían formular como llamadas a un API explícita si queremos

Casos de uso

- Caso de Uso >> Comportamiento >> BDD
- Son ポカヨケ (poka-yoke)
  - Un punto de entrada específico
  - Un lugar de crecimiento claro
  - Fomentan el nivel de abstracción adecuado
- Son la palanca de la complejidad/detalle

¿Hasta dónde llegar con un Caso de Uso?

El nivel de IOC puede ser absurdamente detallado hasta llegar a presentar una capa oculta en el DOM: un form creación de creación oculto y un botón de crear ficha

¿Es la web un detalle?

- Prototipamos antes la web que la lógica de negocio para obtener feedback rápido de los casos de uso
- En servidor es más fácil porque porque la response solo puede ser un fallo, un éxito + redirección o un éxito con un HTML y datos

¿Es la web un detalle?

- Si es un detalle, mi caso de uso debe realizar un control exhaustivo sobre su comportamiento
- Pero una app web es compleja y es frecuente que ante el mismo tipo de estímulo queramos reaccionar de manera distinta
- Esto provocará APIs muy complejas, con funcionalidades parecidas (mucho waste)

Control exhaustivo >> reacciones distintas pero parecidas >> API compleja

El código de la respuesta de un Caso de Uso se parecerá alarmantemente al del Controller web de turno y la API de la web en un pseudo framework web

Nos empeñamos en generalizar para reutilizar pero ¿cómo pretendo reutilizar cuando **no hay dos apps web iguales**?

Si la web no es un detalle…

- Tenemos que tratarla como un hexágono: puertos primarios y secundarios, operaciones sobre la web claras y acotadas, …
- ¿Cómo cambiamos la redacción de los Casos de Uso? ¿Hasta dónde podemos llegar?
- Nuestro cliente/usuario no usa por separado la lógica de negocio y la web, así que ¿cómo garantizo el correcto funcionamiento del conjunto?

Probablemente hay que doblar esfuerzos ya que habrá que **testear** y desarrollar los Casos de Uso tanto de la app como de la web y, aunque por separado son más pequeños, **probablemente se solaparán**.

IOC vs DIP

- La DIP es un subcaso de la IOC
- La IOC es lo que define un framework que invierte el control de tu aplicación.
- El principio Hollywood: “No nos llames, nosotros te llamaremos”
- AngularJS no me llama a mi, yo llamo a AngularJS
- Otro principio de Uncle Bob: El hexágono no depende de AngularJS y viceversa sino que ambos dependen de abstracciones

¿App libre de frameworks?

- No, porque aún necesitamos rellenar carencias de la plataforma JS: AMD, promesas, templating…
- A medida que EcmaScript vaya incorporando todas estas features, ya no serán necesarias estas librerías

RequireJS, Browserify, Q, Mustache, Underscore…

Cosas que nos falta hacer

- Preparar un ejemplo de implementación de Hexagonal con AngularJS con un nivel de IOC bajo, para demostrar el concepto de “la web no es un detalle”

- Estudiar el ejemplo de implementación de Puertos & Adaptadores que se plantea como ejemplo en el GOOS