Els enfocs per escollir un framework

Consultor: Roman Roset roman@uoc.edu

1. Objectius d'aquesta unitat

En aquesta unitat veurem els dos grans enfocs per escollir un framework base de treball. Entre d'altres s' intentarà respondre a aquestes preguntes:

- Cal realment usar un framework ?
- Que és millor Sencha, JQTouch, Jquery Mobile, Jquery, mootols, ...?
- Quines consideracions hem de fer per triar un o l'altre?
- Com podem provar els diferentes frameworks?

Anem a veure tot això!

2. Algunes respostes que hem de tenir clares?

Per què desenvolupem aplicacions d'escriptori amb tecnologia web?

- Perquè volem que una mateixa aplicació sigui multi-dispositiu, i independent del dispositiu tant com es pugui, com passa amb JAVA, o almenys, com passa amb el "C ANSI" en diferents plataformes.
- Perquè coneixem (o estem aprenent) la tecnologia web i volem podem aprofitar els coneixements que ja tenim per a produir.

Quins objectius tenim quan volem fer una aplicació exclusivament per a dispositius mòbils (independentment de si usem el hardware del dispositiu o no)?

- Que les nostres aplicacions es publiquin en un canal de vendes massiu (encara que el producte final tingui cost 0€). Una aplicació en una web no està en cada aparador massiu, i pot quedar aïllada, sense públic.

Quin és l'objectiu no funcionals més important si desenvolupem aplicacions per a mòbil amb tecnologia web?

- Que s'assemblin tant com puguin a una aplicació nativa d'escriptori.

De que depèn que s'assemblin tan com puguin a una aplicació nativa?

1. De poder usar els components del dispositiu si fa falta.
2. D'aprofitar totes les novetats html5:
    a. Que hi hagi geoposicionament web
    b. Que l'aplicació pugui córrer Offline

3. Que la UX (User eXperience) sigui d'escriptori:

    a. en principi un usuari d'aplicacions d'escriptori espera trobar una aplicació d'escriptori. En el fons l'estem "enganyant". L'aplicació correrà sota un navegador web, però ell no ho ha de notar. I això costa, i jo diria que es impossible. De fet... fins a quin punt han de vestir-se totes les aplicacions d'escriptori totes igual? Potser no es un punt en contra sinó un punt a favor.

4. Rendiment: En una aplicació d'escriptori mai s'espera que es carregui una pàgina, com a molt, espera que es carregui de tant en tant informació dins algun element. El rendiment és la clau. S'han de generar les mínimes connexions amb el servidor.

Podeu trobar més informació aquí en un post de Quriksmode que és del 2009, però les característiques que es buscaven són les mateixes. La diferència entre el 2009 i ara, és que el rendiment a millorat moltíssim.

3. Sobre els frameworks

Quin pes li donem a javascript dins la nostra aplicació? En general, per a una lloc web (site), el JS té un pes no superior al 50%, bàsicament és html i css3. En aplicacions més RIA (Rich Internet Application) el pes de Javascript augmenta (google maps p.e.). Fins i tot i han aplicacions on el pes del JS és del 99%: aplicacions d'una sola pàgina, carregues amb AJAX i components definits amb html incrustat en el JS.

Doncs, bé, en general per triar el framework una de les consideracions principals és el pes que té el HTML+CSS en la pàgina o el pes que té el JS.

Actualment s'usen els següents grans enfocs per a construir aplicacions d'escriptori amb JS:

1. Un framework JS principal que fa tota la feina possible (poc control del programador)

Un sant grial que permeti tenir una llibreria de components i opcions, i el desenvolupador és un usuari del framework més que del html, del css i inclús del JS. Ja podeu pensar quins són a nivell conceptual els pros i contres d'aquesta opció. Els grans frameworks que s'usen amb diferents filosofies son:

- Sencha Touch: Sencha Touch és un framework que es basa en aplicacions d'una sola pàgina html (de fet, totes les de mòbil haurien de ser així). I, el que és més espectacular, el framework ho fa tot!! Que volem un component en una pàgina: fem una crida al framework. Que volem carregar les dades en un component: fem una crida al component. Etc... No hi ha html, ja que tot l'html està incrustat en el JS del component i es fica dinàmicament.

- JQuery Mobile: JQuery es una llibreria amb moltes funcionalitats però li dona més pes al html. La filosofia es crear un html5 amb una estructura establerta per a cada component (pàgina, visor, etc..) i automàticament se li associa una lògica amb la llibreria. Per tant, podem treballar amb l'html i el Css de forma amigable i la corba d'aprenentatge és menor. Però, la UX (user eXperince) és pitjor que en el primer.

Podeu trobar més informació i molt útil en aquest enllaços:

- http://markpower.me.uk/workblog/2011/06/sencha-touch-v-jquery-mobile-what-gives
- http://wiki.cetis.ac.uk/images/7/76/Mobile_Web_Apps.pdf
- http://www.quora.com/Were-deciding-between-jQuery-Mobile-currently-in-alpha-and-Sencha-Touch-What-are-the-pros-and-cons-for-each

2. *Un framework que només em doni User Interface (més control)*

Si només volem un framework per a la Interfície gràfica i nosaltres volem encarregar-nos de la resta, aquest enfoc és el nostre. No obstant, estem parlant de encarregar al framework principal TOTA la interfície gràfica.

Aquí la opció més recomanada és JQTouch. Podem veure com programem aplicacions per iPhone amb JQTouch amb aquest tutorial (excel.lent llibre, però del 2009!): http://ofps.oreilly.com/titles/9780596805784/chapAnimation.html

3. *El framework principal el construeixo/defineixo jo. (control total)*

En general aquesta opció seria la que usen els anomenats gurus, que volen un rendiment màxim, una UX única. Aquí hi ha dos punts de vista diferents:

- No s'usa cap framework en particular. Si hi ha la necessitat d'usar un, llavors s'analitza i s´usa en funció del rendiment.
- S'usa una aproximació MVC amb un framework de tamany mínim i rendiment bo. Normalment quan l'aplicació té bastantes pàgines i components a orquestar, no es pot usar el punt anterior i s'opta per aquesta. No obstant, hi ha qui es construeix el seu propi MVC.

D'aquest última opció parlarem en la següent unitat, ja que dona per a un tema en si, i sobretot per a un concepte: el patró arquitectònic Model Vista Controlador en el costat del client, implementat amb JS.

4. *Els empaquetadors*

Un cop funciona l'aplicació executada dintre del navegador, el nostre objectiu és que es pugui engegar sense la necessitat del navegador i així sembli que es tractad'una aplicació nativa 100%. Això ho resolen diferents sistemes com són els següents:

- Sencha Touch: http://docs.sencha.com/touch/2-0/#!/guide/native_android
- Titanium SDK de l'empresa Appcelerator http://www.appcelerator.com
- PhoneGap Es tracta d'un entorn de treball amb eclipse que incorpora una API i un compilador. Normalment usa com a framework base el Cordova-JS

Organitzant una RIA amb JS

Consultor: Roman Roset roman@uoc.edu

4. Objectius d'aquesta unitat

El títol bé podia haver estat: Organitzar una Rich Internet Application amb JavaScript per preparar un bon entorn de desenvolupament, però era massa llarg.

Quan nosaltres volem desenvolupar aplicacions d'escriptori complexes amb tecnologia web per a mòbils entre els nostres objectius no funcionals hauria d'haver (amb més o menys pes) els següents:

- Aconseguir una aplicació d'una sola pàgina html (o molt poques)
- Tenir una estructura de codi molt mantenible i organitzada.
- Automatitzar totes les tasques possibles.
- Intentar fer TDD (test driven development)

Intentarem aconseguir totes aquestes coses amb aquesta unitat, i a més a més, parlar dels frameworks necessaris. Quan en l'anterior unitat anomenava els frameworks "sant grial", el que volia dona a entendre es que, en general, també duent eines per a poder obtenir els objectius que acabo d'anomenar.

1. Aconseguir una aplicació d'una sola pàgina: MVC en JavaScript

Si volem una pàgina html amb molts components, que aquests components es carreguin amb dades i es comuniquin entre ells, ho podem fer amb dos enfocs:

- Enfoc sense organització, ficant el que necessitem i programant sota demanda.
- Enfoc Model Vista Controlador: El model genera crides AJAX al servidor i rep les dades, en la vista hi ha un motor de templates html per a no tenir el html que genera el JS encapsulat amb el JS (Atenció: mai tingueu un JS que incrusta directament html al codi, feu servir templates), el controlador té tota la lògica i monitoritza els events. Molts cops no son veritables controladors sinó, col·leccions.

Cada component (controlador) té la seva vista associada (per exemple una llista) i el seu model (les dades de la llista). Bé, ja coneixeu el model arquitectònic que podeu recordar aquí:

- http://en.wikipedia.org/wiki/Model–view–controller

Per tal d'aconseguir un MVC o bé ho programem des de zero nosaltres mateixos, o bé usem un framework fet. De fet, Sencha i JQuery Mobile usen aquest enfoc, però també moltes més coses. En canvi d'altres frameworks com per exemple backbone o JavascriptMVC, donen els objectes necessaris per a que, a través de extendre el model, creem els nostres propis components. No ens ha quedat clar encara, no? Tranquils, que amb els tutorials de backbone que he afegit, la cosa quedarà més clara d'aquí a 30 minuts.

En aquesta guia us convido a revisar directament backbone, i deixo per vosaltres si també voleu mirar JavascriptMVC. I això per què? Tots dos haurien de funcionar igual de bé o igual de malament per a WAC, però si mai voleu que la vostra aplicació es pugui executar directament des d'un servidor web a través d'internet, llavors, backbone és molt més lleugera "de sèrie" que JavaScriptMVC (que s'ha de "tunejar").

MVC en el costat del client amb Backbone + zetpo + underscore

Us proposo que seguiu el següent recorregut:

1. http://backbonetutorials.com/

Llegiu tots els tutorials "Beginner" de Backbone.

2. Llegiu i proveu aquest tutorial http://thomasdavis.github.com/2011/02/01/backbone-introduction.html
3. Llegiu la pàgina de backbone

Com ho veieu? Doncs ara torneu a llegir el pas 3 (la guia de backbone) i busqueu zepto. Us explico: Podem usar o bé zepto o bé jquery, que són llibreries que entre d'altres donen facilitats als desenvolupadors per navegar i modificar el DOM de la pàgina html. Jquery té moltes més coses que zepto, però no ens calen en general. Per tant, usem zepto:

- Mirar http://mir.aculo.us/2010/10/28/zepto-js-a-jquery-compatible-mobile-javascript-framework-in-2k-presentation/

Ara que ja heu vist *zepto*, o almenys una presentació ràpida, tornem a llegir la pàgina de backbone, i busquem la paraula *underscore*. Underscore ens dona dos coses:

- Aplicar programació funcional [1]
- Un motor de templates [2] i amb mustache.

Per últim us convido a que mireu la següent pàgina:

- http://microjs.com/

i busqueu:

---

[1] Per saber més sobre Programació Funcional en JS: (per a veure que prog. funcional en JS: http://www.ibm.com/developerworks/library/wa-javascript/index.html)

[2] Per veure un exemple de template amb mustache: http://mustache.github.com/#demo

- Un framework base.
- Client-side MVC
- Funcional programin

Com veieu, tot el que em triat es considera un MicroJS.

4. Ara per finalitzar i veure que ho enteneu tot, llegiu aquest tutorial amb atenció: http://arturadib.com/hello-backbonejs/

Fixeu-vos que usa jquery. Podem fer el mateix, però en comptes d'usar jquery podem usar http://zeptojs.com/ que pesa mooooolt menys i té exactament tot el que volem.

2. Tenir una estructura de codi molt mantenible i organitzada.

L'estructura de directoris del projecte és important i en general la tria cadascú (amb excepció de que useu algun framework que la marqui).

Com a mínim tenim la següent estructura, o semblant:

- bin
- css
- js
- Img
- publish

I dintre del js? Doncs ho podem fer tan complicat o senzill com vulgueu. Si usem templates, és important que quedin fora del arxiu de javascript. Si usem MVC i tenim components és útil que cada component tingui el seu directori. Suposem que fem servir un component que és diu taula_llista i un que es diu paginador. Llavors podríem tenir:

- js
    - paginador
        - paginador_model.js
        - paginador_controlador.js
        - paginador_template.html
        - paginador_test.js
    - taula_llista
        - taulallista_model.js
        - blablabla

- lamevaaplicacio.js
- lamevaaplicacio_test.js
- lamevaaplicacio.css

És només un exemple, però crec que s'ha vist la idea. L'organització de la carpeta és important, tant pel manteniment com per a la depuració. Quan depurem hauríem de fer servir el codi de desenvolupar i per tant, podrem fer un bon seguiment del codi.

3. Automatitzar totes les tasques possibles.

S'han d'automatizar tasques? Segurament si, sobretot si volem optimitzar el codi. Però això us ho explicaré en la última unitat d'aprenentatge.

4. TDD: Test Driven Development

Us suggereixo que al programar feu servir una estratègia TDD: Test Driven Development. Es tracta bàsicament de començar la casa per la teulada, vull dir, començar a programar definint unes proves primer i comprovar que totes fallen i anar una per una cadascuna de les proves fent que funcionin. I això dona per a una altra unitat didàctica (petita), que serà la propera.