Disseny i desenvolupament d'una botiga virtual per e-commerce

Alejandro Alonso Gonzalo

Resum—Creació d'una Aplicació web d'una Sola Pàgina, basada en el patró Model-Vista-Controlador i orientada al comerç electrònic. L'aplicació web simularà una botiga on-line relacionada amb el món del cinema, on es podrà comprar contingut cinematogràfic. Aquest nou concepte, més ràpid i eficient, fuig del paradigma de desenvolupament habitual de la web on la majoria de la lògica de negoci es realitza a la banda del servidor. Ara, serà el client qui s'encarregui de realitzar una gran part de la lògica de negoci aprofitant els seus recursos i alliberant de càrrega al servidor. El desenvolupament de l'aplicació a la banda del client es basa en AngularJS, un framework Javascript de codi obert i sostingut per l'empresa Google. AngularJS va néixer a l'any 2009 però que no ha sigut fins l'any 2013 quan s'ha fet del tot popular. A més a més, es fa especial èmfasi en el tema de la seguretat. S'han de tenir en compte possibles usuaris malintencionats que desitgen explotar vulnerabilitats existents al sistema. Xifratge de les comunicacions o prevenció d'atacs externs entre d'altres, seran algunes de les solucions que s'ofereixen per mantenir protegit d'amenaces el sistema.

Paraules clau—aplicació web, comerç electrònic, patró Model-Vista-Controlador, Aplicació d'una Sola Pàgina, framework AngularJS, seguretat.

Abstract— We are going to create a single page application based on Model-View-Controller pattern. The electronic commerce application, commonly known as e-commerce simulates an on-line cinema shop where the customer can buy cinematographic content. This new concept, more fast and efficient, avoids the usual paradigm of web development where most of the business logic is performed on the server side. Now, the customer will be responsible for performing the most of the business logic, leveraging their resources and releasing the server load. The application development on the client side is based on AngularJS. AngularJS is an open source JavaScript framework which is supported by Google. It was born in 2009 but it was not until 2013 when it has been quite popular. Also, we are going to emphasize with security. We need to pay special attention with malicious users who wish to exploit existing vulnerabilities in the system. Encryption of communications and preventing external attacks are some of the solutions offered to keep the system protected from threats.

Index Terms—web application, e-commerce, Model-View-Controller pattern, Single Page Application, AngularJS framework, security.

1 INTRODUCCIÓ

AVUI dia, la tecnologia es troba a tot arreu. És com una mena de motor que mou el món. Fa 50 anys, ningú podia imaginar el creixement que ha tingut fins ara. I és que no només utilitzem la tecnologia en l'àmbit social, d'oci i/o entreteniment, sinó que també ho fem en l'àmbit comercial i/o de negocis. Fa uns anys, era inimaginable que qualsevol usuari, des de casa seva, pogués comprar un televisor a una botiga situada a l'altra punta del món, sense moure's.

Per mitjà d'una enquesta [1] realitzada per l'Institut Nacional d'Estadística durant l'any 2013, el 69,8% de les llars a Espanya disposen de connexió a Internet. Pel que fa el comerç, un estudi [2] realitzat a finals de l'any 2013, mostra com un 59% dels usuaris de la Unió Europea han fet compres per Internet durant l'any 2012. És a dir, gairebé 6 de cada 10 persones ja utilitzen de forma habitual Internet per adquirir tot tipus de productes des de qualsevol dispositiu (ordinador, tableta o telèfon mòbil).

Amb les dades al davant, podem veure que Internet ha esdevingut una font de negoci molt important. En els últims anys, hem pogut observar com totes aquelles empreses que no s'han adaptat als canvis tecnològics han acabat reduint molt la seva producció, fins i tot, arribant a tancar el seu negoci per falta d'ingressos. La introducció de les *websites* i més endavant, del comerç electrònic (també conegut com *e-commerce*), ha fet que el mercat fos encara més competitiu. D'altre banda, moltes d'aquelles empreses que van fer un esforç i van invertir per adaptar-se a la nova era tecnològica, han vist un gran creixement (tant econòmic com mediàtic) del seu negoci, com indica a [3].

L'article està format per les següents seccions: la primera secció, defineix i classifica els objectius del projecte, a més de presentar l'estat de l'art.

La segona secció, parla de la metodologia de desenvolu-

- *E-mail de contacte: alejandro.alonsog@e-campus.uab.cat*
- *Menció realitzada: Tecnologies de la Informació.*
- *Treball tutoritzat per: Ramon Musach Pi (departament d'Enginyeria de la Informació i de les Comunicacions).*
- *Curs 2013/14*

pament que s'ha fet servir durant tota aquesta etapa, per dur a terme el projecte. També es mostra un llistat dels diferents requeriments del sistema (funcionals, no funcionals i restriccions).
La tercera secció, revela els diferents patrons de *software* que s'han aplicat al projecte, així com, la seva importància dintre del desenvolupament d'una aplicació web. En aquesta mateixa secció, també es fa una pinzellada sobre l'arquitectura de la aplicació i les tecnologies que s'han emprat.
La quarta secció, fa especial èmfasi en el *framework* AngularJS, tecnologia en la que es fonamenta l'aplicació web a la banda del client. A més d'explicar els conceptes bàsics, es fa una petita introducció de les estructures de dades que s'han fet servir al projecte.
La cinquena secció, fa referència a les SPA (Aplicacions d'una Sola Pàgina o *Single-Page-Applications*). Com funcionen i quines avantatges tenen respecte al paradigma de desenvolupament d'aplicacions web que s'ha seguit fins ara.
La sisena secció, tracta la seguretat de l'aplicació web i de les dades de caràcter personal que es troben emmagatzemades al servidor. Seguretat en les comunicacions entre el client i el servidor o defensa d'atacs externs, són alguns dels temes a tractar.
La setena secció, fa una breu explicació sobre com adaptar la nostre aplicació web als diferents dispositius mòbils i sistemes operatius.
La vuitena secció, mostra els resultats obtinguts en el projecte després d'aquest llarg període d'implementació.
Finalment, a la novena secció s'exposen les conclusions i es suggereixen possibles millores de cara al futur.

1.1 Objectius

El principal objectiu és dissenyar i desenvolupar una aplicació web orientada al comerç electrònic (botiga online o virtual) basada en el patró MVC (Model-Vista-Controlador). L'aplicació web, seguirà un nou paradigma de desenvolupament web anomenat SPA (*Single Page Applications* o Aplicacions d'una Sola Pàgina). La botiga, estarà relacionada amb el món del cinema i la televisió. Qualsevol usuari registrat, tindrà accés a tot el contingut de la pàgina web i podrà realitzar les seves compres emplenant el seu carro virtual amb les pel·lícules i/o series que més li agradin. L'usuari també podrà valorar els diferents productes que hi ha en estoc.
Un altre objectiu, és aplicar mesures de seguretat a la nostre aplicació web, per tal de defensar-nos sobre possibles atacs externs. La *web app* contindrà informació sensible (dades personals dels clients) que cal protegir aplicant la "Llei orgànica de protecció de dades de caràcter personal (15/1999)". Farem servir certificats digitals per tal de xifrar les comunicacions entre el client i el servidor. També prendrem mesures amb alguns dels atacs més comuns com per exemple: SQL Injection, XSS (*Cross Site Scripting*) o XSRF (*Cross Site Request Forgery)* i vulnerabilitats JSON (*JavaScript Object Notation*).
Per últim i com a objectiu menys important, adaptarem la nostre aplicació a dispositius mòbils o *tablets* per tal que es pugui accedir de forma còmoda i senzilla amb altres dispositius que no siguin l'ordinador. Els dispositius mòbils són una eina que cada cop s'utilitza més i que la majoria de persones porten a sobre al llarg del dia. D'aquesta manera, no serà necessari un ordinador per poder visualitzar de forma clara i ordenada el contingut de la *web app*.
A la taula I, es troben llistats i classificats els objectius del projecte. Cada objectiu és classificat com a crític, prioritari o secundari depenent del seu grau d'importància dins el projecte. En total, hi han sis objectius. Dos d'aquests objectius són crítics, tres són prioritaris i només hi ha un que és secundari.

TAULA I
LLISTAT I CLASSIFICACIÓ D'OBJECTIUS

Objectiu	Crític	Prioritari	Secundari
Dissenyar i desenvolupar una aplicació web orientada al comerç electrònic	X		
Utilitzar el patró Model-Vista-Controlador		X	
Adaptar l'aplicació a versió mòbil			X
Xifrar les comunicacions entre client i servidor		X	
Protegir d'atacs externs les dades de l'usuari		X	
Finalitzar el projecte en el temps determinat	X		

1.2 Estat de l'art

Una aplicació web o *web app*, és una aplicació *software* que s'emmagatzema a un servidor remot i que és accessible a través d'un navegador o client web per mitjà d'Internet i el protocol HTTP (*Hypertext Transfer Protocol*).
Per una banda, el client web o navegador és l'encarregat de gestionar totes les peticions de l'usuari i de rebre els documents en format HTML (*Hypertext Transfer Markup Language*) que provenen del servidor.
Per un altre banda, el servidor és l'encarregat d'esperar peticions per part d'un client i respondre'l quan escau amb els documents HTML i els recursos necessaris.
El protocol HTTP permet definir la sintaxi i la semàntica que han de seguir tant el client com el servidor, per tal de comunicar-se de forma eficient.
La popularitat de les aplicacions web, esdevé ja que es pot accedir per mitjà de qualsevol client web, amb independència del sistema operatiu i sense la necessitat per part de l'usuari d'haver d'instal·lar cap *software* al sistema. A més a més, és fàcil d'actualitzar i mantenir al llarg del temps.
Encara que ens pugui sonar estrany, són moltes les aplicacions web que coneixem i que utilitzem al llarg del dia. Aquí tenim alguns exemples:

- Webmails.
- Weblogs.
- Wikis.
- Botigues on-line.

És molt habitual, accedir al correu electrònic per mitjà de clients de correu o *webmails* que proveeixen una interfície web a l'usuari. Alguns dels clients de correu més populars són: "Microsoft Outlook", "Mozilla ThunderBird" o "eM Client".
També són coneguts els *blogs* o *weblogs*, on normalment un autor publica texts o articles i una sèrie de lectors li fan arribar la seva opinió.
Altres tipus de aplicacions web són les *wikis*, on diferents usuaris poden crear, modificar o eliminar texts informatius a través del navegador web, i les botigues on-line, també conegudes com botigues virtuals o botigues electròniques. Aquest tipus d'aplicacions es basen en el comerç a través d'Internet.
En el meu cas, l'aplicació web es tracta d'una botiga on-line. La motivació principal per triar aquest projecte, ha sigut la possibilitat d'aprofundir en l'aprenentatge de tecnologies emergents en el desenvolupament d'aplicacions web. Per altre banda, la complexitat del projecte i el seu ús futur per part d'usuaris reals del sector tecnològic, han fet que sigui un repte especialment atractiu per a mi.

2 METODOLOGIA

Per dur a terme aquest projecte, he decidit utilitzar la metodologia de desenvolupament en espiral. La metodologia de desenvolupament en espiral, és una evolució de la metodologia de desenvolupament en cascada, també coneguda com *waterfall*. Aquesta mètode de desenvolupament és considerat iteratiu i incremental. Iteratiu ja que es van fent iteracions sobre un mateix procés per tal d'obtenir el màxim desitjat, i incremental, perquè es van afegint funcionalitats a mesura que augmenta el nombre d'iteracions.
El seu objectiu, és fallar d'hora i aprofitar al màxim possible el *feedback* que rep a cada iteració. D'aquesta manera, s'aconsegueixen esmenar errors en etapes poc avançades del projecte, que permeten disminuir el risc d'inconsistència a etapes futures del projecte.

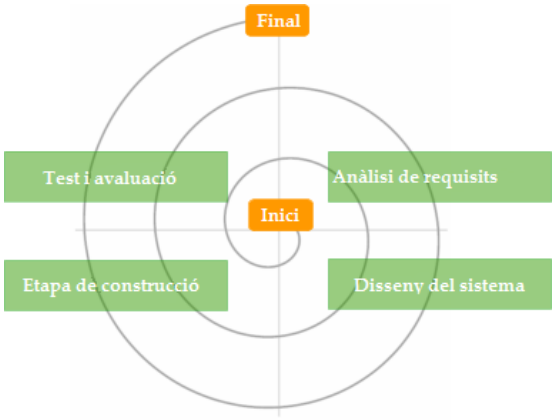

Fig. 1. Etapes de la metodologia de desenvolupament en espiral.

Com s'observa a la Fig. 1, la metodologia de desenvolupament en espiral consta bàsicament de quatre etapes.
A la primera etapa, es fa un anàlisi en profunditat dels requisits. La segona etapa és la etapa de disseny. La tercera etapa és la etapa de codificació i per últim, la etapa de test i avaluació. Aquest cicle es va repetint contínuament i es van generant cada cop versions més completes i complexes del producte. Quan s'arriba a l'objectiu final desitjat, s'atura el procés.
Personalment, crec que aquesta metodologia és la més adient per aquest tipus de projecte per diversos motius:

1. És un mètode de desenvolupament ràpid i eficient.
2. Els objectius finals estan definits, però no sabem detalladament la seva implementació.
3. Permet agafar *feedback* constant del client, ajudant a adaptar de manera continua els requisits inicials.
4. S'aconsegueixen funcionalitats bàsiques de forma ràpida, augmentat la motivació de l'equip.
5. Els recursos del projecte es mantindran constants.

Per assolir tota la llista d'objectius, s'hauran de determinar una sèrie de requisits funcionals, no funcionals i restriccions.

2.1 Requisits funcionals

- L'usuari ha de poder registrar-se.
- L'usuari ha de poder fer *login*.
- L'usuari ha de poder veure les diferents categories de productes.
- L'usuari ha de poder veure les característiques d'un determinat producte.
- L'usuari ha de poder valorar un producte.
- L'usuari ha de poder filtrar i/o ordenar una llista de productes.
- L'usuari ha de poder afegir, eliminar o editar productes al carro de la compra.
- L'usuari ha de poder realitzar una compra.
- El sistema ha de permetre modificar les dades de compra d'un usuari.
- L'usuari ha de poder veure el progrés de compra.
- El sistema ha de permetre a l'usuari restablir la seva contrasenya.
- El sistema ha d'estar protegit enfront atacs externs.
- El sistema ha de xifrar les comunicacions.

2.2 Requisits no funcionals

- L'aplicació web ha d'ésser multi plataforma.
- Les comunicacions han d'ésser xifrades per mitjà d'un certificat digital amb OpenSSL.
- El disseny del sistema haurà d'ésser seguint patrons de disseny.

2.3 Restriccions

- Una direcció de correu electrònic no pot estar vinculada a més d'un compte.
- Un usuari no registrat no pot realitzar una compra.
- Un usuari no registrat no pot valorar un producte.

3 PATRONS I DISSENY D'INTERFICIES

A l'hora de crear l'aplicació web, he fet especial èmfasi en l'ús de patrons, tant d'arquitectura com de disseny d'interfícies. Els patrons són molt útils, ja que són solucions ben pensades a problemes de programació molt coneguts. L'ús de patrons ajuda a: evitar buscar una solució de forma reiterada d'un problema en concret, formalitzar un vocabulari comú entre tota la comunitat de programadors, estandarditzar certs comportaments o facilitar l'aprenentatge de les noves generacions. Simplement es tracta de no reinventar la roda.
Cal dir, que tampoc és aconsellable abusar dels patrons ni fer-ne un ús excessiu.
Per una banda, s'ha fet servir el patró d'arquitectura de *software* conegut com MVC (Model-Vista-Controlador) [4]. Aquest patró, ens permet separar les dades de l'aplicació (model), la interfície d'usuari (vista) i la lògica de negoci (controlador) en tres components, com veiem a la Fig. 2.

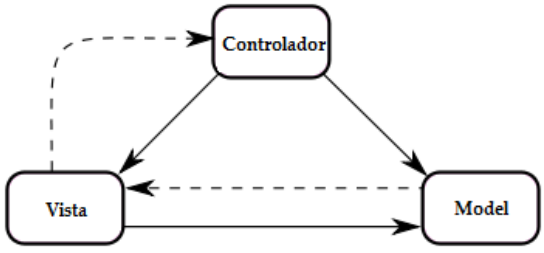

Fig. 2. Patró d'arquitectura *software* MVC (Model-Vista-Controlador).

Això pot semblar no gaire útil, però en aplicacions web amb una certa dimensió, és essencial pel bon funcionament. Aquesta distinció entre model, vista i controlador, ens permet fer canvis en qualsevol de les tres parts de forma independent sense cap tipus de dependència entre mòduls. Per tant, tenim un sistema altament escalable i preparat per qualsevol tipus de canvi, sense que això comporti una gran remodelació en tota la estructura de l'aplicació.
Per exemple, en un moment determinat podríem passar d'una base de dades relacional a una no relacional a la banda del model, sense que això afectés en el funcionament general de l'aplicació.
Per un altre banda, he fet servir patrons de disseny d'interfícies. Amb la lectura d'alguns capítols de [5] he aprés cert tipus de patrons de disseny que es troben a la majoria d'aplicacions web actuals i que serveixen per donar bones sensacions a l'usuari. Igual que un disseny lleig no ens agrada i fa que fugim ràpidament d'una pàgina web, una mala funcionalitat per culpa de no haver utilitzat cap patró de disseny pot portar a situacions similars.
Un clar exemple és el disseny de formularis. Quan un usuari es vol registrar a la teva aplicació web, el primer que fa és emplenar tots els camps del formulari amb les seves dades personals. Un mal disseny en aquest cas, seria no informar en cap cas a l'usuari de si les dades que està introduint són correctes o no. Molts cops, l'usuari també s'emprenya quan clica el botó de "registrar-se" i al haver introduït alguna dada de forma incorrecte, el sistema li buida tots els camps del formulari i li retorna l'error. L'usuari ha de tornar a emplenar tots els camps del formulari, per un error del que no era conscient.
Un altre exemple, és el que s'anomena com *scent* o petjada. Un bon disseny d'una aplicació web és visible quan un usuari sap en tot moment on es troba i pot accedir amb pocs clics a qualsevol lloc de la web. Quan un usuari fa clic masses vagades a la fletxa per tornar enrere del navegador, vol dir que el disseny de l'aplicació no és del tot correcte.
Exemples com aquests, es solucionen aplicant patrons de disseny. És molt important aplicar aquests patrons ja que millorem l'autoestima del client i augmenten la seva fidelitat. Per petits errors com aquests, es perden un munt de clients que en definitiva, són un munt de calers.

3.1 Arquitectura de l'aplicació web

Un dels punts més importants del projecte és decidir quina serà l'estructura del projecte, així com quines tecnologies es faran servir [6].
A la Fig. 3, podem observar una petita representació del que és l'arquitectura de l'aplicació web.

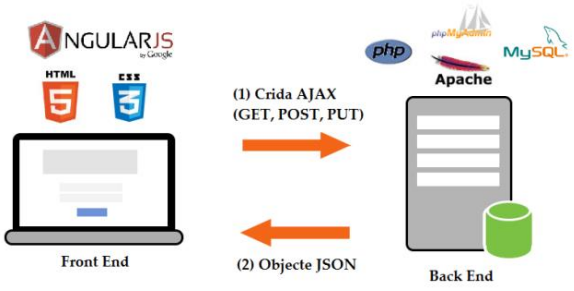

Fig. 3. Arquitectura de l'aplicació web.

Com qualsevol arquitectura web, es tracta d'una arquitectura client-servidor. Per una banda, a la banda del client (*front end*) s'han utilitzat bàsicament les següents tecnologies:

- HTML5: és l'última versió d'HTML (*Hypertext Transfer Markup Language*), llenguatge de programació que s'utilitza per desenvolupar pàgines web. La informació s'organitza per mitjà d'etiquetes (també conegudes com *tags*), que indiquen al navegador com ha de representar el contingut de la pàgina web. És un estàndard a càrrec de la W3C.
- CSS3: és l'última versió de CSS (*Cascading Style* Sheets). La seva aplicació més comú, és per descriure la presentació semàntica (format i aspecte) d'un document escrit en llenguatge HTML.
- AngularJS: és un *framework* Javascript opensource per desenvolupar aplicacions web utilitzant el patró MVC (Model-Vista-Controlador). És molt flexible i de fàcil lectura. Està desenvolupat per la empresa Google i sembla que 2013 és l'any del seu enlairament.

Per un altre banda, a la banda del servidor (*back end*) s'ha instal·lat LAMP. LAMP és un acrònim que s'origina per descriure la plataforma sobre la que funcionen les aplicacions web (Linux, Apache, MySQL i PHP). La combinació de totes aquestes tecnologies defineix la infraestructura d'un servidor web, utilitzant un paradigma de programació per el desenvolupament. A continuació, podem vuereu una petita descripció de cadascuna d'aquestes tecnologies:

- Apache: és un servidor web HTTP de codi obert i multiplataforma (Unix, Windows o Macintosh).
- PHP: és un llenguatge de programació interpretat que s'executa a la banda del servidor. S'utilitza especialment per modificar dinàmicament el contingut de pàgines web. Actualment, s'incrusta dintre del codi HTML de pàgines web. Permet establir connexions amb bases de dades i disposa de versions tant per a Windows com per a Linux.
- MySQL: és un sistema de gestió de bases de dades relacional multi-fil i multiusuari, que utilitza el llenguatge SQL (Structured Query Language). S'ha fet molt popular gràcies a la seva velocitat en executar consultes en l'elaboració d'aplicacions web, en l'entorn del programari lliure.
- phpMyAdmin: és una eina en llenguatge PHP que permet administrar bases de dades MySQL a través d'un client web o navegador. Facilita el treball de modificar, eliminar, o afegir dades a la base de dades (molt visual).

La comunicació entre *front end* i *back end* és realitza mitjançant objectes de tipus JSON (*JavaScript Object Notation*). JSON és un format senzill i que pesa poc, molt utilitzat per l'intercanvi de dades.
Per tant, quan *front end* necessiti per exemple, un llistat dels productes de la categoria "cinema en tres dimensions", farà una crida AJAX GET al servidor i aquest li retornarà en format JSON un objecte amb tots els productes de la categoria "cinema en tres dimensions".
Com he comentat anteriorment, aquesta estructura ens permet per exemple, canviar a la part de *back end* la utilització de PHP com a llenguatge de programació per un altre com podria ésser nodeJS. Sempre i quan, el servidor en nodeJS retornés un objecte de tipus JSON a la banda de *front end*, tot continuarà funcionant perfectament, sense necessitat de cap canvi a *front end.*
A la banda de *front end*, he utilitzat aquest conjunt de tecnologies pensant en poder assolir l'objectiu de construir una aplicació web multi plataforma, adaptada als diferents tipus de dispositius (ordinador, tableta o telèfon mòbil). Una aplicació web, és una web en la que s'accedeix a través d'una URL mitjançant el navegador del dispositiu i que s'adapta al format de la pantalla.
Existeixen altres tipus d'aplicacions com són les aplicacions nadives o les aplicacions híbrides. Aquests tipus d'aplicacions, tenen una resposta més ràpida i consumeixen menys recursos que les aplicacions web. D'altra banda, són més costoses de fer.
Pel que fa la banda del *back end,* he triat la pila LAMP (Linux, Apache, MySQL i PHP) per raons d'experiència i temps. Pensant en el poc temps que tenia, no he volgut endinsar-me en noves piles com podrien ésser MEAN (MongoDB, Express, AngularJS i NodeJS) on s'utilitza MongoDB com a SGBD (Sistema Gestor de Base de Dades), Express com a *framework* per gestionar el servidor i NodeJS com a servidor web JavaScript.

4 FRAMEWORK ANGULARJS

AngularJS [7] és un *framework* de codi obert, orientat al desenvolupament d'aplicacions web a la banda del client, també conegut com *front end.* AngularJS utilitza el llenguatge de programació Javascript i està basat en el patró MVC (Model-Vista-Controlador). Va ésser desenvolupat per la empresa Google a principis de l'any 2009 i el seu creixement està essent molt gran en aquests moments. Tant és així, que en els últims anys s'està consolidant com un dels *frameworks* Model-Vista-Controlador més utilitzat en l'entorn de les aplicacions web. AngularJS permet crear SPA (*Single Page Applications* o Aplicacions d'una Sola Pàgina), provocant en l'usuari una experiència més propera. La informació detallada de les SPA es troba al punt 7 d'aquest mateix article.
Com qualsevol tecnologia, AngularJS té les seves avantatges i els seus inconvenients. Les avantatges del seu ús són:

1. És una tecnologia emergent.
2. Ha estat desenvolupat per Google (suport continu).
3. Sistema *data binding*.
4. Gran part de la lògica de negoci es troba a la banda del client.
5. Gran expectació (gran comunitat de *developers*).

Alguns dels seus inconvenients són:

1. Documentació oficial escassa.
2. Quedes lligat a ell (com qualsevol altre *framework*).
3. Corba d'aprenentatge abrupta.

4.1 Client-side template

En molts dels *frameworks* pel desenvolupament d'aplicacions web que coneixem fins ara, el servidor és l'encarregat de barrejar les dades amb els *templates* (plantilles) en format HTML i enviar tot aquest contingut al navegador per tal que aquest pugui generar la vista.
Aquest concepte és completament diferent a AngularJS. A AngularJS, el servidor proporciona *templates* estàtics. Per un altre banda, el client és l'encarregat de generar la vista, barrejant les dades (que provenen del model) amb els *templates* HTML proporcionats pel servidor anteriorment.
Amb això, s'aconsegueix aprofitar una gran quantitat de recursos disponibles a la banda del client, que en aquest moment determinat romanen lliures. D'aquesta manera s'allibera una gran part de recursos a la banda del servidor, que permetrà treballar a aquest amb molta més agilitat i velocitat.

A causa de la gran capacitat de càlcul de la majoria de màquines d'avui dia, aquest mètode no suposa cap problema pel client (que no s'adona de que la seva màquina està treballant una mica més que anteriorment) i facilita molt la feina al servidor.

4.2 Data binding

Una de les característiques més destacades de AngularJS, és la coneguda com *data binding* o sincronisme a temps real entre la vista i el model. Ara, cada controlador treballa amb un tipus de variable que s'anomena $scope. Per mitjà d'aquesta variable, es crea una sincronització bidireccional entre la vista i el model.
Això, provoca que qualsevol canvi que es realitza en la vista, influeix de manera directa en el model. De la mateixa manera, qualsevol canvi que es fa al model es reflexa a la vista. Una de les grans avantatges d'aquesta particularitat pròpia d'AngularJS, és que permet fer canvis a la vista a temps real i tot això, sense necessitat de refrescar la pàgina web. Aquesta particularitat, és coneguda com two-way data binding.
A la Fig. 4, podem veure un diagrama on es veu el flux d'execució.

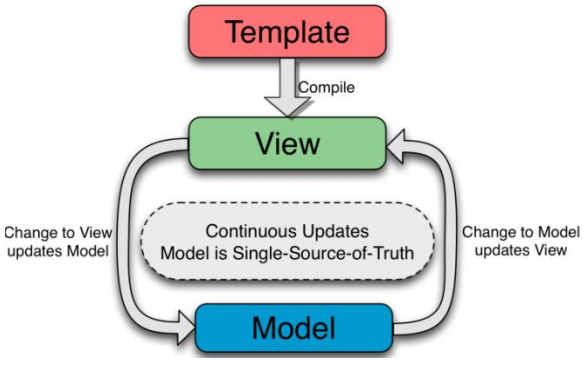

Fig. 4. Flux d'execució *two-way data binding*.

Un altre de les avantatges del *data binding*, és que disminueix de manera notable el número de peticions al servidor. Això succeeix, ja que una vegada es fa la petició al servidor i es guarda la resposta en la variable $scope, el client pot interactuar amb la vista, provocant canvis continus en aquesta. Davant d'aquesta situació, el client creu que s'estan realitzant noves crides al servidor , quan realment no és així i el que realment està succeint, és que s'està filtrant la informació de sortida per pantalla (vista) amb un filtratge sobre la variable $scope.
Com he comentat en el punt 4.1, això permet no sobrecarregar de forma continua el servidor, cada cop que el client interactua amb la vista. A més a més, permet mantenir una comunicació constant i a temps real amb el client, que la fa molt atractiva.

4.3 Directives i filters

Una altre de les característiques rellevants d'AngularJS, és l'ús de *directives* (directives) i *filters* (filtres).
Una directiva és simplement, un marcador sobre un element de DOM (*Document Object Model*), com podria ser una classe CSS (*Cascading Style Sheets*). L'ús de directives dins de tags HTML, afegeix noves funcionalitat i comportaments, és a dir, estén HTML. AngularJS inclou una llarga llista de directives, encara que també és possible crear noves directives per a usos concrets.
Totes les directives comencen amb el prefix "ng". Aquí tenim alguns exemples: "ng-app", "ng-controller", "ng-submit", "ng-click", "ng-repeat", "ng-show" o "ng-hide".
En el següent exemple, queda una mica més clar l'ús de les directives.
A la Fig. 5, es veu l'estructura del controlador, amb un variable $scope declarada amb el nom "*cities*" en format JSON. En aquesta variable es guarda el nom d'un conjunt de ciutats com són París o Madrid.

```
var myApp = angular.module('myApp',[]);

myApp.controller('mainController', function($scope) {
    $scope.cities = [{name:"París",name:"Madrid"}]
});
```

Fig. 5. Controlador amb la variable "*cities*" declarada.

A la Fig. 6 s'observa la vista HTML amb la directiva "ng-repeat" inserida dins del tag "ul". El que permet aquesta directiva concretament, és iterar sobre la variable "*cities*" permetent imprimir el valor del camp "name" per cada element de l'array.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>AngularJS exemple ng-repeat</title>
</head>
<body ng-app="myApp">
    <ul ng-controller="mainController">
        <li class="list" ng-repeat="city in cities">
            {{city.name}}
        </li>
    </ul>
</body>
</html>
```

Fig. 6. Vista HTML amb amb la *directive* "ng-repeat" al tag "ul".

D'altra banda, els filtres tenen la funció de filtrar la informació abans d'imprimir per pantalla. La seva implementació és molt similar al de les directives. El gran avantatge de l'ús de filtres, és que no necessites fer noves trucades al servidor per seleccionar la informació desitjada, sinó que, a partir d'un conjunt d'informació, es pot aplicar un filtre per tal de restringir allò que podrà ser vist en la vista.

4.4 Services i factories

Normalment, cada vista està associada a un controlador.
AngularJS, disposa de dos estructures de dades anomenades *services* (serveis) i *factories* (factories), que permeten definir una sèrie de funcions i variables, que més tard poden ésser injectades en cadascun dels controladors existents. Aquest tipus de estructures són molt eficients i molt utilitzades en AngularJS. A continuació, veurem un exemple per veure molt més clar el seu ús.
La aplicació web, que ja vàrem comentar que està relacio-

nada amb el món del cinema, té quatre categories de productes (pel·lícules en dvd, pel·lícules en Blu-ray, pel·lícules en tres dimensions i series).
Cada categoria de productes, té la seva vista HTML i el seu controlador associat (per tant, disposem de quatre vistes i quatre controladors). A més de mostrar tota la informació relacionada d'un producte, un producte pot ésser afegit al carro de la compra per mitjà del botó "afegir item al carro de la compra".
Això vol dir, que en un primer instant, cadascun dels quatre controladors hauria de disposar de una funció "afegir item" amb tota la lògica per tal de permetre al usuari fer clic a qualsevol producte de qualsevol categoria i que aquest, fos afegit al carro de la compra.
En aquest cas només tenim quatre categories, però imagineu una web amb cinquanta o cent categories.
Davant aquest problema de redundància, apareixen els *services* i els *factories*.
A la Fig. 7, podem veure un exemple d'un *service*. En aquest cas, es tracta d'un *service* implementat per cobrir les funcionalitats del carro de la compra (inicialitzar el carro de la compra, afegir un producte, eliminar un producte, fer un recompte del total de productes, actualitzar una quantitat o buidar el carro).

```
myApp.service('cartService', function($cookieStore) {

    var cart = [];

    return {
        init: function() {
            if (!$cookieStore.get('cart')) {
                $cookieStore.put('cart', []);
            }
        },
        getCart: function() {
            return $cookieStore.get('cart');
        },
        addItem: function(item, quantity) {
            ...
        },
        deleteItem: function(item) {
            ...
        },
        updateQuantity: function(item, quantity) {
            ...
        },
        emptyCart: function() {
            ...
        }
    };
});
```

Fig. 7. Service del carro de la compra.

A partir d'aquí, qualsevol controlador, de qualsevol de les vistes associades a una categoria de productes, pot injectar aquest *service* i accedir a les funcions desenvolupades en aquest.
En la Fig. 8, el controlador associat als productes de tipus Blu-ray, injecta "cartService" (s'observa a la part superior) i accedeix a la funció de afegir un producte al carro de la compra.
Si ens fixem atentament, la funció "addItem" que es troba al controlador no és la més "interna", sinó que és la funció que crida al *service* (que és on es troba tota la lògica).

```
myApp.controller('dvdController', function($scope, cartService) {

    //Clic al botó d'afegir producte al carro
    $scope.addItem = function(item, quantity) {
        cartService.addItem(item, quantity);
    };

});
```

Fig. 8. Injecció de "cartService" al controlador Blu-ray.

5 APLICACIONS D'UNA SOLA PÀGINA (SPA)

Per tal de millorar la fluïdesa de la *web app* i l'experiència de l'usuari, s'ha implementat el paradigma SPA (*Single Page Application* o Aplicació d'una Sola Pàgina). El paradigma SPA permet que tot el contingut d'una aplicació web s'ajusti en una sola pàgina. Es defineix una estructura base (index) en format HTML, amb un tag de contingut que anirà variant segons el flux de l'aplicació. Això es coneix com *injection content*.
L'estructura de la *web app* a la banda del *front end* es veu reflexada a la Fig. 9.

```
- script.js          <!-- codi AngularJS -->
- index.html         <!-- template base -->
- pages              <!-- pàgines html -->
----- home.html
----- login.html
----- dvd.html
```

Fig. 9. Estructura de l'aplicació web a la banda del *front end*.

Disposem d'un *template* base (index.html), on es troba l'estructura principal de la web. Per altra banda, tenim tot el conjunt de vistes (fitxers en format HTML) on només haurem de definir el seu *tag* amb l'identificador "container".

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>AngularJS SPA</title>
</head>
<body ng-app="myApp" ng-controller="mainController">
    <header>
        <!-- header -->
    </header>
    <div id="container">
        <!--injecció de contingut-->
        <div ng-view></div>
    </div>
    <footer>
        <!-- footer -->
    </footer>
</body>
</html>
```

Fig. 10. Estructura del fitxer "index.html".

Amb aquest mecanisme, aconseguim que la part estàtica de la web no s'hagi de carregar cada cop que l'usuari canvia de vista al navegador (alliberant així molts recursos), sinó que tenim una part estàtica que només es carregarà un sol cop i un *tag* dinàmic (que s'anomena en aquest cas "container") que serà el que anirà variant de contingut segons el flux de la *web app*. Per una banda, l'usuari no se'n adonarà de que l'únic que canvia és el *tag* "container" i per altra banda, la web proporcionarà molta

més fluïdesa i velocitat de transició a l'usuari.
A la Fig. 10, podem veure quina és la estructura del fitxer "index.html" i en quin *tag* es produeix la *injection content* de la resta de pàgines HTML.

5.1 Routing i templating

Un cop s'ha definit l'estructura SPA, s'ha de configurar el fitxer "script.js" com mostra a la Fig. 11. Aquest fitxer tindrà una relació vista-controlador on s'accedirà cada cop que l'usuari faci clic en una vista determinada de la pàgina web. Per exemple, si l'usuari fa clic al botó que li porta a la vista de pel·lícules en dvd "/dvd", per mitjà d'aquest fitxer el servidor sap que ha de carregar el *template* que s'anomena "dvd.html" que es troba a la carpeta "pages" i a més a més, que aquest *template* va lligat amb un controlador que s'anomena "dvdController".
Si d'altre banda, l'usuari fa clic al botó que li porta a la vista de *login* "/login", el servidor carregarà el *template* "login.html", que es troba a la carpeta "pages" i el seu controlador corresponent, anomenat "loginController".
La velocitat de càrrega de qualsevol de les vistes HTML utilitzant el paradigma SPA, és molt més superior que utilitzant el paradigma habitual de desenvolupament, on per cada vista es torna a carregar tot el *template* HTML sencer.

```
var myApp = angular.module('myApp', ['ngRoute']);

    //Configuració de les diferents rutes
    myApp.config(function($routeProvider) {
        $routeProvider

            //Ruta per la pàgina principal
            .when('/', {
                templateUrl : 'pages/home.html',
                controller  : 'mainController'
            })

            //Ruta pel login
            .when('/login', {
                templateUrl : 'pages/login.html',
                controller  : 'loginController'
            })

            //Ruta pels productes dvd
            .when('/dvd', {
                templateUrl : 'pages/dvd.html',
                controller  : 'dvdController'
            });

            ...
    });
```

Fig. 11. Estructura del fitxer "script.js".

6 SEGURETAT

La quantitat de dades personals que s'emmagatzemen en els sistemes d'informació, ha anat creixent al llarg del segle XXI. Cada cop més usuaris maliciosos intenten burlar les barreres de seguretat de les aplicacions web, per fer-se amb tota aquesta informació i treure profit econòmic o social. Les aplicacions en general, i les aplicacions web en particular, acostumen a tindre errors de programació que poden arribar a afectar en la seva seguretat.
A continuació, s'explica com protegir el sistema enfront alguns dels atacs més coneguts en l'àmbit de les aplicacions web.

6.1 Xifratge de les comunicacions

Per tal de xifrar les comunicacions entre client i servidor, el primer que s'ha fet ha sigut treballar amb OpenSSL (*Open Source Toolkit for SSL/TLS*) [8] per tal de crear un certificat digital auto signat (accés protegit HTTPS). OpenSSL és un projecte de software lliure que consisteix en un ampli paquet d'eines criptogràfiques i que ens permet crear un certificat digital signat per nosaltres mateixos com a CA (Autoritat de Certificació). En aquest cas, al principi de la barra de direccions (URL) es troba el cadenat que indica que existeix un certificat digital. Sortirà de color vermell, ja que el propi navegador (que té una llista de CA's contrastades) no detecta la nostra CA entre el seu llistat i avisa al client de que aquesta *web app* no és segura. Normalment, les empreses amb un cert prestigi accedeixen a CA contrastades (com per exemple Symantec [9]) per tal d'obtenir el seu certificat digital que assegura la seva identitat. Nosaltres, com no tenim calers per fer això, el que hem fet és fer-nos passar per una CA i signar-lo nosaltres mateixos.

6.2 SQL Injection

L'atac SQL Injection és un dels més comuns dintre de l'entorn de les aplicacions web.
La immensa majoria d'aplicacions web, emmagatzemen tota la informació en SGBD (Sistema Gestor de Base de Dades) amb suport pel llenguatge SQL.
Aquesta informació sol estar protegida per diferents mecanismes que proporcionen un nivell de seguretat suficient, inclús, sistemes d'autenticació i controls d'accés.
Tot i així, aquestes mesures de seguretat poden ésser sortejades per un usuari maliciós, aprofitant una mala pràctica a l'hora de programar diferents components.
Per tal de protegir l'aplicació enfront aquest tipus d'atacs, existeixen varies alternatives:

1. Consultes parametritzades o *Parametrized queries*.
2. Procediments emmagatzemats o *Stored procedures*.
3. Escapament de les dades d'entrada o *Escape inputs*.
4. Assignar els privilegis mínims o *Least privilege*.
5. Validació de les dades d'entrada amb llista blanca o *White list input validation*.

En aquest projecte s'ha aplicat la primera de les alternatives possibles. *Parametrized queries*, consisteix en precompilar una instrucció SQL per tal de separar, per una banda les dades d'entrada del usuari i per l'altre, la pròpia instrucció SQL. D'aquest mode, s'evita que el SGBD confongui els *inputs* del usuari amb el codi SQL de la consulta i sigui possible una injecció de codi maliciós que alteri el comportament del SGBD.
També és possible utilitzar la funció "mysql_real_scape_string" de la biblioteca de MySQL (que pertany al grup d'escapament de les dades d'entrada) per evitar aquest tipus d'atac. De totes mane-

res, és molt més recomanable acostumar-se a programar utilitzant consultes parametritzades que haver de posar sempre aquest tipus de funció, ja que moltes vegades i sense ser conscients, oblidem posar-la i tornem a deixar un forat de seguretat.
Tota la informació sobre SQL Injection la he tret de [10], un llibre molt interesant del autor Chema Alonso, un dels referents en seguretat informàtica més importants del món.

6.3 Cross Site Request Forgery (XSRF)

XSRF és una tècnica per la qual un lloc no autoritzat pot obtenir les teves dades personals i privades.
Al contrari que en els atacs de XSS els quals exploten la confiança que un usuari té en un lloc concret, el XSRF explota la confiança que un lloc té en un usuari concret.
Per contrarestar aquesta vulnerabilitat, AngularJS proporciona un cert mecanisme.
Quan es fa una petició XHR (XMLHttpRequest), el servei $http llegeix un *token* que es troba a la *cookie* anomenat XSRF-TOKEN i ho defineix com una capçalera HTTP (X-XSRF-TOKEN). Atès que només JavaScript s'està executant en el teu domini, podria llegir la *cookie*. El servidor, podria estar segur de que la petició XHR s'està executant en el teu domini.
Per dur a terme aquest mecanisme, el servidor necessita definir un *token* a la *cookie* de la sessió JavaScript anomenat CSRF-TOKEN, quan es fa la primera crida HTTP GET. Tot seguit, XHR pot verificar que la *cookie* coincideix amb la capçalera X-CSRF-TOKEN HTTP i estar segur de que només el codi JavaScript que s'està executant en el teu domini, ha pogut enviar la petició HTTP.
El *token* generat ha d'ésser únic per cada usuari. Ademés, ha d'ésser verificable pel servidor per tal d'evitar que JavaScript s'estigui generant els seus propis *tokens.*

6.4 Vulnerabilitat JSON

Si l'aplicació web retorna objectes de tipus JSON, s'ha de tenir en compte que poden haver possibles forats de seguretat. Aquesta vulnerabilitat, permet que una pàgina es vegi compromesa per un atac de XSS aprofitant la sessió que té oberta el client en una tercera part de confiança.
Si s'envia una petició a un servei, el qual et retorna un objecte de tipus JSON, llavors és possible recuperar dades d'un tercer domini on anteriorment el client ha iniciat sessió. Això és molt perillós, ja que proporciona un accés potencial a dades de tipus privat. Com els objectes de retorn són de tipus JSON, és fàcil de parsejar com una sentencia JavaScript.
A continuació explicaré un breu exemple per que quedi més clar.
Imaginem que hem iniciat sessió a www.elmeubanc.es, un servei que retorna objectes de tipus JSON. Més tard, navegant per internet i sense voler, entrem al domini www.jsonattack.com on existeix un script malintencionat que accedeix a les dades de tipus JSON que retorna www.elmeubanc.es, a través de la sessió que has iniciat anteriorment. D'aquesta forma i sense adonar-nos, estem proporcionant informació sensible al atacant.
Per protegir el sistema davant d'aquest tipus d'atac, s'ha de afegir una petita cadena de caràcters ")]}',\n" com a prefix de cada petició JSON. AngularJS estriparà automàticament aquesta cadena de caràcters abans de processar-la com un JSON qualsevol.

7 DISPOSITIUS MÒBILS

7.1 PhoneGap

PhoneGap [11] és un *framework* gratuït i multiplataforma que permet desenvolupar aplicacions mòbils multiplataforma a partir de tecnologies web com podrian ésser HTML5, CSS3 i JavaScript. PhoneGap funciona sobre Apache Cordova i va ésser creat per Nitobi. Actualment, el *software* és propietat de la empresa Adobe.
La aplicació mòbil generada per PhoneGap és un tipus d'aplicació híbrida, és a dir, es situa entre el conjunt d'aplicacions nadives i el conjunt d'aplicacions purament web.
Una de les grans avantatges d'aquest *framework*, és que permet exportar la aplicació a una sèrie de sistemes operatius sense necessitat d'haver d'aprende cada llenguatge natiu en profunditat. Alguns dels sistemes operatius que suporta són: iOS, Android, Windows Phone, BlackBerry OS. Això, ens permet crear una aplicació web fàcilment exportable a dispositius mòbil, sense haver de contar amb personal especialitzat en cadascun dels sistemes operatius pels quals volem adaptar l'aplicació.
En el moment que la empresa és adquirida per Adobe, surt a la llum PhoneGap Build. PhoneGap Build, és una plataforma on-line que permet compilar la teva aplicació sense necessitat de tenir un MAC per compilar la teva aplicació per iOS, un Windows per compilar la teva aplicació per Windows Phone, etc...
A la Fig. 13, podem veure el seu funcionament.
PhoneGap Build et permet pujar una aplicació de forma gratuïta. A partir d'aquí, les següents aplicacions que vulguis compilar al núvol són de pagament.
És un servei molt fàcil d'utilitzar i que s'està sobreposant a la metodologia habitual de programar en llenguatge natiu per cada sistema operatiu.

8 RESULTATS

En termes generals, els resultats finals del projecte han sigut bons. S'han assolit amb èxit la gran majoria dels objectius marcats a l'inici, en el temps establert.
Els objectius s'han anat abordant segons el seu nivell d'importància dintre del context del projecte. Primer els objectius crítics i en última instancia els objectius secundaris.
En primer lloc, s'ha assolit amb èxit el principal objectiu que era dissenyar i desenvolupar una aplicació web orientada al comerç electrònic en quatre mesos. S'ha aconseguit desenvolupar tant la part de *front end* com la part de *back end* amb èxit, incorporant nous paradigmes de desenvolupament com les Aplicacions d'una Sola Pàgina o *Single Page Applications*.
En segon lloc, s'ha assolit amb èxit l'objectiu de desenvolupar l'aplicació web seguint el patró Model-Vista-Controlador, fent servir tecnologies emergents com és el

cas d'AngularJS.
En tercer lloc i no menys important, s'ha assolit amb èxit la tasca de protegir la *web app* enfront possibles atacs externs d'usuaris maliciosos. S'ha dut a terme una cerca dels

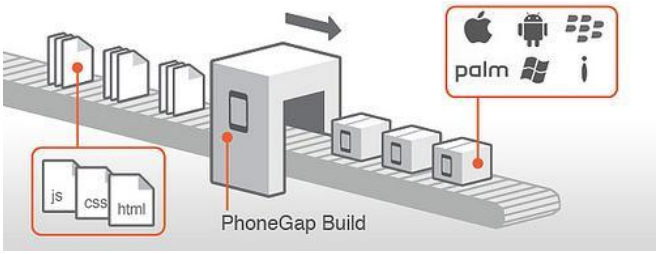

Fig. 13. Funcionament de PhoneGap Build.

atacs més comuns dintre del sector de les aplicacions web i s'han implementat certs mecanismes per protegir l'aplicació.
Tot aquest conjunt fa que sigui una aplicació ràpida, escalable i oberta a canvis.
Per últim, l'únic objectiu que no s'ha pogut completar ha sigut el d'adaptar l'aplicació per dispositius mòbils.
La incorporació d'AngularJS i la nul·la experiència amb aquest nou *framework*, varen fer que no hi hagués temps suficient per completar al cent per cent aquest objectiu.
Com s'explicava anteriorment, aquest era un objectiu secundari i per tant, no ha afectat en cap moment al correcte desenvolupament del projecte.

9 CONCLUSIONS

A partir del treball realitzat en tot aquest temps puc treure una sèrie de conclusions i propostes de millora. Com he comentat anteriorment, la meva idea principal era assolir tots els objectius i adaptar l'aplicació web a dispositius mòbils amb l'eina PhoneGap. Degut a la immersió dintre del món AngularJS i la seva complicada corba d'aprenentatge, no ha sigut possible dur-ho a terme en aquests quatre mesos. He preferit perfeccionar la versió web i afegir més funcionalitats, que donar pas ràpidament a la implementació de la versió mòbil.
Al principi del projecte, ja n'era conscientment del poc temps que tenia i per tant, ja vaig definir aquest objectiu com objectiu secundari. Com he definit al punt 1.1, era la tasca que considerava menys rellevant dintre del meu projecte.
Per una banda, he aconseguit dissenyar i desenvolupar una aplicació web, utilitzant a la banda del client una tecnologia (AngularJS) en plena expansió dintre de l'entorn web. Per altre banda, he après a augmentar el nivell de seguretat de la *web app* enfront alguns dels atacs externs més populars. Per últim, he après com funciona el desenvolupament d'aplicacions mòbils multi plataforma.
En aquesta vida tot és millorable, per tant, aquest projecte també és millorable en alguns aspectes. Algunes de les millores que crec que farien encara millor l'aplicació serien les següents:

1. Implementació d'una API per fer la *web app* RESTful (per exemple amb Jersey).
2. Implementació del servidor amb nodeJS (faria un projecte molt més escalable).
3. Adaptar la *web app* a versió mòbil.
4. Internacionalització. Adaptar la *web app* a diferents llengües (anglès, català o alemany).
5. Implementació d'una passarel·la de pagament.

He de dir que estic molt content amb el meu treball i amb tot l'esforç que hi he dedicat. Crec que han sigut 4 mesos molt durs però molt satisfactoris. Per tant, em dono per satisfet amb el treball realitzat i amb tot el que he après en aquests 4 mesos de treball intens.

AGRAÏMENTS

M'agradaria donar les gràcies en primer lloc, a la Universitat Autònoma de Barcelona per donar-me la possibilitat de realitzar aquest treball. En segon lloc, m'agradaria donar les gràcies al meu tutor Ramon Musach Pi, pels seus consells, el seu suport continu i la seva implicació durant tots aquests mesos.
Per últim, donar les gràcies especialment a la meva família. El meu pare Benito, la meva mare Paula i el meu germà Adrián. Han sigut uns mesos molt durs en els que sempre he rebut el seu recolzament i m'han ajudat a tirar endavant en tots els moments difícils.

BIBLIOGRAFIA

[1] Instituto Nacional de Estadística. Encuesta sobre Equipamiento y Uso de Tecnologías de Información y Comunicación en los Hogares (TIC-H), 2013. [Consulta: 20 juny 2014]. Disponible a: http://www.ine.es/prensa/np803.pdf
[2] Diari El Mundo. España, lejos de Europa a la hora de hacer compres por Internet, 2013. [Consulta: 20 juny 2014]. Disponible a: http://www.elmundo.es/elmundo/2013/10/15/economia/1381836578.html
[3] Diari El País. El comercio electrónico en España coge impulso en el primer trimestre, 2013. [Consulta: 20 juny 2014]. Disponible a: http://economia.elpais.com/economia/2013/09/10/actualidad/1378807154_017751.html
[4] Department of Computer Scienceat Illinois. How to use Model-View-Controller (MVC). [Consulta: 20 juny 2014]. Disponible a: http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html
[5] Jenifer Tidwell. Designing Interfaces. 2ª ed. O'Reilly.
[6] W3C (WorldWide Web Consortium). [Consulta: 25 maig 2014]. Disponible a: http://w3schools.com/
[7] AngularJS. [Consulta: 20 juny 2014]. Disponible a: https://docs.angularjs.org/tutorial
[8] OpenSSL: The Open Source Toolkit for SSL/TLS. [Consulta: 20 juny 2014]. Disponible a: http://www.openssl.org/
[9] Symantec. Presentación de SSL, 2013. [Consulta: 20 juny 2014]. Disponible a: http://www.symantec.com/es/es/theme.jsp?themeid=how-ssl-works
[10] Enrique Rando, Chema Alonso. Hacking de Aplicaciones Web: SQL Injection. 2ª ed. 0xWord.
[11] PhoneGap. [Consulta: 20 juny 2014]. Disponible a: http://phonegap.com/