UNIVERSITAT DE BARCELONA

Treball final de grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques
Universitat de Barcelona

MyWall: aplicació RESTful dissenyada per a socialitzar l'escalada en rocòdroms

---

Autor: Víctor Oriol i Aguilar

Director: Dr. Eloi Puertas Prats
Realitzat a: Departament de Matemàtiques aplicades i anàlisi

Barcelona, 27 de gener de 2016

Abstract

The choice of this project was motivated by the union of two passions, information engineering and climbing, both of them completely different character.
The project has a target completely innovative and ability to reach the entire community of climbers who practise indoor climbing. This community has been increasing for the last 15 years and now results a new market to be exploited.

The main objective of the project is to bring a traditional old sport to a new and present platform as a mobile application.

This application has an intuitive interface which is easy to use and it allows anyone with basic knowledge on how to use a smartphone rune it.
To start using the application we only need a phone with Android OS and Internet connection (3G, 4G or WiFi).

The result of the project is an application oriented to the users of an indoor rock climbing gym so they can check on the different available climbing routes and score whenever they finish each of them. This score will result in a ranking of users and it will increase competitiveness among them.
Concurrently, a light web interface for the fitness centre administrators wash scheduled in order for them to manage this network of sportspeople.

Resum

L'elecció d'aquest projecte va ser motivada per la unió de dues passions, la informàtica i l'escalada, ambdues de caràcter completament diferent.
El projecte té un objectiu del tot innovador i la capacitat d'arribar a tota la comunitat d'escaladors que practiquen aquest esport en sala tancada, ja que aquesta ha anat en augment des d'els últims 15 anys, creant un mercat nou i per explotar.

L'objectiu principal del projecte és portar un esport antic i tradicional a una plataforma actual, com és una aplicació mòbil.

Aquesta aplicació compta amb una interfície molt intuïtiva i fàcil d'utilitzar, i així permet que qualsevol persona amb coneixements bàsics en l'ús d'un 'smartphone' pugui fer-la servir.
Per començar a utilitzar l'aplicació només és necessari un mòbil amb SO Android i connexió a Internet (3G, 4G o Wifi).

El resultat del projecte és una aplicació orientada als usuaris d'un gimnàs on es practica l'escalada que permetrà veure les diferents rutes disponibles i anotar quan s'ha finalitzat una, generant un rànquing de socis i augmentant així la competitivitat entre els escaladors.
Paral·lelament s'ha programat una lleugera interfície web pels administradors del gimnàs capaç de gestionar aquesta xarxa de socis.

En l'escalada, el cervell és el múscul més important - Wolfgang Gullichs

Índex

1 Introducció

Es considera que l'esport de l'escalada neix l'any 1786 amb l'ascensió al Mont Blanc per el Dr. Gabriel Paccard i el seu guia Jacques Balmat.
Després de diversos anys d'evolució, als anys 50 neix el Boulder, modalitat nascuda en els boscs de Fontainebleau (França). L'inici del segle XXI es caracteritza per una popularització de l'escalada esportiva a tots els nivells socials, donant lloc, entre altres, a l'obertura de rocòdroms, instal·lacions preparades específicament per practicar l'escalada amb l'objectiu d'evitar desplaçar-se a la muntanya.

Per tant, l'objectiu principal d'aquest projecte és aplicar les noves tecnologies a aquest esport en auge i sense explotar des del punt de vista informàtic.

Paral·lelament s'ha programat una lleugera interfície web pels administradors del gimnàs capaç de gestionar aquesta xarxa de socis.
Es permetrà l'inserció, eliminació i modificació de socis i blocs. Tots els canvis s'aplicaran de forma automàtica i instantània en l'aplicació mòbil.

1.1 Definició d'objectius

L'objectiu principal d'aquest projecte és crear una lliga orientada a gimnasos on es practica l'esport de l'escalada, on cada soci pugui anotar les vies que finalitza amb èxit, indicant el nombre d'intents que ha necessitat. En funció de la dificultat de la via i el nombre de vegades que s'hagi provat el problema, l'escalador obtindrà una puntuació que serà sumada al seu total.

També es vol crear un apartat de 'favorits' que donarà lloc a una lliga privada i facilitarà el seguiment dels teus companys dintre de l'aplicació.

A més, es vol idear un sistema per a poder accedir als blocs mitjançant codis QR, de tal manera que serà possible arribar a un bloc des del llistat general o amb la càmera del mòbil.

1.2 Motivació del problema

La motivació d'aquest projecte és la unió de dues passions, la informàtica i l'escalada, ambdues de caràcter completament diferent.

Com a escalador amb més de 10 anys d'experiència he provat les diferents branques d'aquest esport, moltes com és el muntanyisme (trekking) o alpinisme, vies d'escalada clàssica o de diversos llargs dificulten la informatització de l'activitat, donat que es fan en terrenys molt abruptes i amb condicions que dificulten l'ús de telèfons mòbils.
En canvi, la pràctica d'escalada en una sala tancada és un entorn ideal per a l'ús d'aplicacions adaptades a un 'smartphone'

Com a soci d'un dels millors rocòdroms ubicats a la ciutat de Barcelona, on entrenen professionals d'aquest esport, he pensat crear una petita lliga inter-rocòdrom per augmentar l'esperit de competició entre socis.

1.3 Estructura de la memòria

El següent document intentarà explicar al complet el projecte 'MyWall'. Començarem definint el marc de treball de l'aplicació i explicant alguns conceptes sobre escalada i tecnologia de la informació.
Després donarem un cop d'ull al mercat actual per veure si la nostra aplicació hi té lloc. En acabat, parlarem de la viabilitat, tant tecnològica com econòmica, del projecte.
Un cop vist que el treball és comercial i viable analitzarem el problema que volem resoldre i explicarem com solucionar-ho des d'un punt de vista de disseny.
A continuació explicarem com s'ha desenvolupat el projecte: Instal·lació de l'entorn de treball, servidor i serveis, implementació de l'aplicació mòbil i finalment l'aplicació web.
Un cop vist què hem fet i com ho hem fet passarem a analitzar els resultats obtinguts, per nosaltres i per possibles clients finals.
En acabat conclourem el projecte i parlarem d'algunes millores que es podrien implementar en un treball futur.

Ho veiem tot a continuació.

2 Marc de treball i conceptes previs

Donat que els objectius del treball són els de crear una aplicació per a entorns d'escalada indoor, introduirem el marc de treball i conceptes previs relacionats amb aquests dos mons. Començarem parlant sobre l'escalada, què és i d'on ve.

2.1 Definició d'escalada

L'escalada és l'acció d'ascendir per un indret difícil, de pregona verticalitat, utilitzant els quatre membres per progressar i consta de múltiples disciplines.

A continuació farem un breu repàs del pas de l'escalada en la història.

2.1.1 Breu història de l'escalada

Es considera que l'esport de l'escalada neix l'any 1786 amb l'ascensió al Mont Blanc per el Dr. Gabriel Paccard i el seu guia Jacques Balmat, a partir d'aquest moment i amb l'evolució dels materials, condicionant aquests en gran manera la progressió i seguretat en les activitats, se segueix una línia cap a la cerca del repte esportiu i la dificultat donant lloc a objectius verticals e inaccessibles.

En els anys 20 es desenvolupa àmpliament l'escalada lliure, que limita les ascensions a l'ús de peus i mans com a mètode de progressió. Posteriorment, als anys 50 neix el Boulder (o escalada en bloc), modalitat nascuda en els boscs de Fontainebleau (França), que consisteix en l'escalada de roques, habitualment de menys de 6 metres d'alçada, sense cordes ni aparells d'escalada. L'objectiu final de les vies és que l'escalador arribi al capdamunt del bloc de roca.
Més tard, els anys 80 porten amb ells el naixement de l'escalada esportiva, es tracta de l'ascensió de parets exclusivament en lliure i utilitzant la corda com a assegurança. Aquesta és la disciplina més practicada avui en dia. (veure figura 1)

L'inici del segle XXI es caracteritza per una popularització de l'escalada esportiva a tots els nivells socials, donant lloc, entre altres, a l'obertura de rocòdroms, instal·lacions preparades específicament per a practicar l'escalada amb l'objectiu d'evitar desplaçar-se a la muntanya.
A continuació aprofundirem en l'escalada orientada a sales d'entrenament, ordinàriament anomenades rocòdroms, donat que és l'escenari principal d'aquest projecte.

Figura 1: Timmy O'Neil i Dean Potter abans del seu record de velocitat en The Nose - El Capitan a Yosemite (2001).

2.1.2 Escalada orientada a sales d'entrenament (rocòdroms)

Un rocòdrom és una instal·lació preparada específicament per practicar l'escalada amb l'objectiu d'evitar desplaçar-se a la muntanya. Està equipat amb preses, assegurances i reunions, donant lloc a diferents vies d'escalada. La seva forma i mida son lliures tot i estar condicionades per la forma de l'edifici on s'ubiqui.
Existeixen sales on es practica l'escalada esportiva, l'escalada en bloc o ambdues.

- Preses: Objectes de diferents grandàries, formes i colors, que simulen els agarris que es poden trobar en una paret de muntanya.

- Assegurances: Son petites plaquetes d'acer inoxidable utilitzades en la pràctica d'escalada esportiva, s'han d'ancorar mitjançant un cargol de la mateixa estructura del rocòdrom. Durant l'ascensió, l'escalador s'encordarà en aquestes plaquetes per disminuir el factor de caiguda i evitar tocar el terra.

- Reunions: Les vies d'escalada esportiva solen disposar d'una assegurança al final del recorregut, formada normalment per una anella i un mosquetó d'acer soldat amb tancament de filferro.

- Via: Conjunt de preses que formen un recorregut i, en el cas de l'escalada esportiva, finalitza en una reunió. (veure figura 2)

Figura 2: Sala de boulder deudits a Barcelona

Un cop contextualitzada la temàtica de l'aplicació passarem a comentar alguns conceptes previs de caràcter tecnològic.

2.2 Conceptes previs

Per la implementació d'aquest projecte s'han utilitzat diverses tecnologies, a continuació passarem a explicar les més específiques.

2.2.1 Aplicació WEB

En l'enginyeria de programari s'anomenen aplicació web les aplicacions que els usuaris poden emprar accedint a un servidor web a través d'Internet o d'una intranet amb un navegador. En altres paraules, és una aplicació de programari que es codifica en un llenguatge suportat pels navegadors web i es confia l'execució al navegador.

Les aplicacions web s'han popularitzat per la senzillesa del navegador web com a client lleuger, la facilitat per actualitzar i mantenir aplicacions sense distribuir i instal·lar programari a milers d'usuaris potencials, poder accedir a la informació personal des de qualsevol dispositiu connectat a Internet i treballar sobre una mateixa aplicació diversos usuaris.

2.2.2 Aplicació mòbil

Les APPs (aplicacions mòbils) són programes que s'instal·len en un dispositiu mòbil i que es poden integrar a les característiques tècniques del dispositiu. Proporcionen accés instantani a un contingut específic sense haver de buscar-lo a Internet. A més, es poden actualitzar per afegir noves característiques amb el pas del temps.

Una aplicació per a dispositius mòbils es manté en el telèfon i en la tauleta de l'usuari i és, per tant, idònia per la seva utilització freqüent i repetida. En conseqüència, respon a una necessitat específica.

2.2.3 URL

L'URL (Uniform Resource Locator, localitzador uniforme de recursos) és una adreça formada per caràcters alfanumèrics que indica la localització d'un fitxer o d'un directori a Internet i que permet accedir-hi.

2.2.4 JSON

JSON (acrònim de JavaScript Object Notation) és un estàndard obert basat en text dissenyat per a l'intercanvi de dades i llegible per humans. Deriva del llenguatge script JavaScript, per representar estructures de dades simples i llistes associatives, anomenades objectes.
El format JSON s'utilitza habitualment per a 'serialitzar' i transmetre dades estructurades en una connexió de xarxa. S'utilitza principalment per intercanviar dades entre un servidor i una aplicació web, essent una alternativa a l'XML.

JSON està constituït per dues estructures:

- Una col·lecció de parelles de nom/valor.

- Una llista ordenada de valors. (veure figura 3)

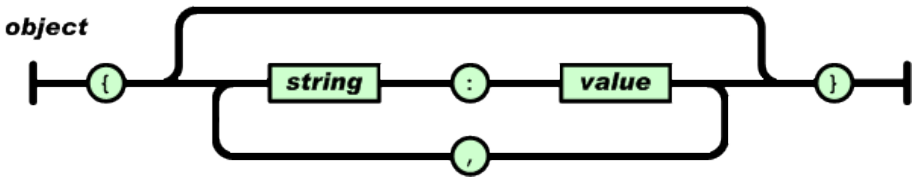

Figura 3: Estructura d'un objecte JSON

Les dades enviades entre l'aplicació mòbil i el servidor web han estat codificades en format JSON .

2.2.5 QR

El codi QR (Quick Response) és un sistema per emmagatzemar informació en una matriu quadrada de punts dissenyada per ser llegida amb la càmera d'un telèfon intel·ligent o tauleta tàctil (entre d'altres).
La informació codificada en una etiqueta QR conté caràcters alfanumèrics que poden contenir text simple, adreces web i targetes de presentació en format Vcard (entre d'altres). (veure figura 4)

Figura 4: Codi QR que codifica la URL: http://37.59.100.31:8080/ranking/bloc/00007/

S'han utilitzat els codis QR per poder accedir mitjançant la càmera del telèfon intel·ligent a un bloc.

Fins ara hem vist la temàtica de l'aplicació i hem introduït alguns conceptes tecnològics per entendre una miqueta més aquest projecte, a continuació es parlarà de l'estudi de mercat i viabilitat, tecnològica i econòmica, que ha impulsat el projecte 'MyWall'.

3 Estudi de mercat i viabilitat

Les aplicacions relacionades amb el món de l'escalada disponibles al mercat Android es poden dividir en 4 grans grups. (veure figura 5)

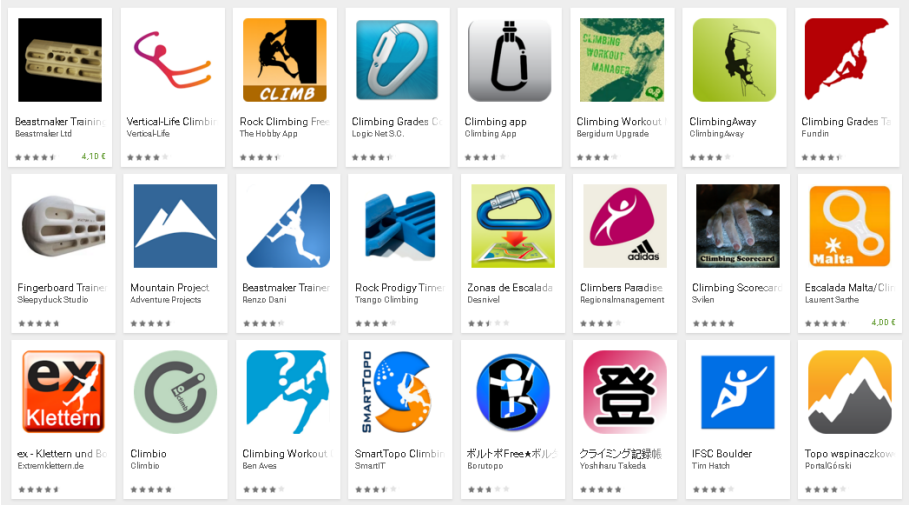

Figura 5: Aplicacions disponibles al mercat d'Android per a escaladors

1.- Entrenament personal: Son aplicacions orientades a l'entrenament personal; com a norma general inclouen exercicis de preparació física específics per millorar el rendiment de l'escalador, un comptador de repeticions i un cronometre. Ocupen el 30% del mercat.

2.- Convertidors de grau: La dificultat d'una via d'escalada es mesura en graus, i cada regió té la seva pròpia notació, les més utilitzades són l'Americana i la Francesa, els espanyols utilitzem aquesta última. Aquestes aplicacions funcionen com un convertidor de moneda. Ocupen el 10% del mercat.

3.- Ressenyes: No existeix una base de dades oficial amb les vies d'escalada equipades, indicant el seu grau i localització, és la mateixa comunitat d'escaladors qui manté aquestes dades de forma dispersa i desatesa. Existeixen diferents aplicacions amb la seva pròpia base de dades que permet als usuaris documentar els accessos i el nombre de vies dels diferents sectors. Ocupen el 30% del mercat.

4.- Rànquings: Tot esport té una faceta competitiva, l'esport de l'escalada no és menys i des dels últims 15 anys aquesta ha anat creixent fins al punt de proposar l'escalada com a esport olímpic. Per tant no és d'estranyar que existeixi un mercat d'aplicacions que permetin a l'usuari anotar les vies que finalitza amb èxit donant lloc a petites lligues d'escalada. Aquest grup ocupa l'altre 30% del mercat.

L'aplicació 'MyWall' forma part del 4rt grup però amb un caràcter completament innovador i una nova filosofia.

Com s'ha explicat anteriorment, des de l'inici dels temps de l'escalada, tota la informació relativa a accessos, graus i localització de les vies ha estat mantinguda per la mateixa comunitat d'escaladors, donant lloc a una disgregació total del coneixement en aquest sector. Com a conseqüència és molt difícil fer una lliga global a nivell amateur, donat que és difícil comparar quelcom del que no es té una informació 100% certa o validada.
Amb l'objectiu d'esmenar aquest problema neix 'MyWall'. Una aplicació on les dades es troben ordenades i reglades per professionals del sector, donant lloc a una base de dades coherent i actualitzada i, en conseqüència, una lliga reglada i estable. Un altre factor que ha incentivat aquest projecte és que no s'ha trobat cap gimnàs on es practiqui l'esport de l'escalada amb una aplicació pròpia, però sí la necessitat de tenir-la, donant lloc a un nou i gran mercat per explotar.

Després d'estudiar el mercat d'aplicacions actual i confirmar que l'aplicació que es vol dissenyar té lloc i futur ha sigut necessari efectuar un estudi de viabilitat tecnològica i econòmica per saber si és o no viable impulsar el projecte 'MyWall', a continuació es comenten els resultats d'aquests estudis.

3.1 Viabilitat tecnològica

La quota de mercat Android va consolidar el seu lideratge en 2014, arribant al 83,3% de la quota de mercat, i al 84.7% en 2015 gràcies a l'aposta d'Android pels telèfons intel·ligents de baix preu. Per aquest motiu s'ha decidit basar aquest projecte en Android.

Per a que aquest projecte sigui possible és necessari un servidor que allotgi una base de dades, una aplicació web que ens serveixi les dades en un format que puguem tractar des de l'aplicació mòbil i un enton de test, sigui un telèfon mòbil intel·ligent amb sistema operatiu Android o un emulador (Android Studio).
Després d'estudiar el mercat tecnològic actual s'han trobat moltes tecnologies open-source i d'estàndards oberts que podrem fer servir per a dur a terme el nostre projecte.

S'ha decidit utilitzar el següent programari en el costat del servidor per a servir les dades a l'aplicació mòbil:

- Sistema operatiu: Centos 7, és un clon a nivell binari de la distribució GNU/-Linux Red Hat Enterprise Linux RHEL, compilat per voluntaris a partir del codi font alliberat per Red Hat.

- Base de dades: MariaDB, és una branca del sistema de gestió de bases de dades MySQL impulsada per la comunitat, per tal de mantenir el seu estat lliure sota la GNU GPL.

- Aplicació WEB: Django Framework, és un framework de desenvolupament web de codi obert escrit en Python.

La viabilitat tecnològica de l'aplicació 'MyWall' és possible gràcies al conjunt d'eines de software lliure i protocols oberts utilitzats que, entre altres coses, ofereixen molta documentació i suport als programadors. S'ha de tenir en compte també que no és necessari un 'hardware' i software' especial per a consumir el producte.

Ara que sabem que a nivell tecnològic és possible desenvolupar la nostra idea, es requereix fer un estudi econòmic, pel fet que, que una idea sigui bona no vol dir que sigui rentable. Veiem els resultats a continuació.

3.2 Viabilitat econòmica

El 'hardware' i 'software' indispensable per poder consumir 'MyWall' és un telefon intel·ligent amb SO Android, com hem explicat en el punt anterior. Actualment, aquest ocupa el 84.7% del mercat. Per tant, en la majoria dels casos no és necessària inversió econòmica per part del client o soci.

El servidor on s'allotja la base de dades i el servidor web contractat al proveïdor de hosting OVH té un cost de 2,4e mensuals (iva inclòs). Es considera assumible.

El software utilitzat per dur a terme aquest projecte és open-source, sumant un cost de 0e al projecte en llicències.

Per tant, la viabilitat econòmica de l'aplicació 'MyWall' és possible ja que és un producte de baix cost i accessible al públic.

Un cop tenim clar que el desenvolupament del nostre projecte és possible a nivell tècnic i econòmic podem entrar en la fase d'anàlisi de requisits i disseny. Ho comentem a continuació.

4 Anàlisi i Disseny

En aquesta secció es parlarà de què volem fer i com ho volem fer. Començarem parlant del problema que volem resoldre per a després explicar com ho farem.

Com hem explicat anteriorment, es pretén dissenyar i programar una aplicació mòbil per a telèfons intel·ligents amb sistema operatiu Android orientada a sales d'escalada 'indoor'.

Els usuaris que consumiran aquesta aplicació no tenen un perfil concret, donat que a dia d'avui en un rocòdrom pots trobar des de nens de 4 anys fins a adults de 70, tant nois com noies practicant aquest esport en la mateixa sala i intentant resoldre els mateixos blocs.
El que aquestes persones tenen en comú és la passió per aquest esport i que compten amb totes les facultats físiques per a utilitzar una aplicació mòbil, per tant no és necessari tenir en compte barreres d'aquest tipus durant el disseny de l'aplicació.

L'aplicació disposa d'un llistat de socis i blocs.
El llistat d'usuaris ve ordenat per punts, on el capdavanter de la llista és el soci amb la puntuació més alta dintre del rocòdrom i l'últim el que té la més baixa.
Com els blocs de la sala canvien, el llistat de blocs està ordenat per data d'obertura, posant sempre els deshabilitats al final.
Es disposa d'un tercer llistat, el de favorits, un soci pot crear un llistat d'usuaris favorits donant lloc a una lliga personal, aquest llistat també es troba ordenat per puntuació.

Al polsar sobre un bloc trobarem l'opció de sumar-lo a la nostra puntuació i un llistat ordenat per data dels usuaris que han finalitzat aquesta via. També disposa de l'opció de veure els comentaris d'altres competidors.
Al polsar sobre un soci podrem veure tots els blocs que ha finalitzat i la seva imatge de perfil (en cas de tenir-ne una).

El projecte també disposarà d'una pàgina web per a que els empleats del rocòdrom pugin gestionar els diferents usuaris i blocs. Tots els canvis que s'executin des d'aquest portal seran accessibles des de l'aplicació mòbil al moment.
Aquesta pàgina serà simple però intuïtiva, podrem escollir entre mostrar el llistat de blocs o d'usuaris i un cop dintre de la secció es podrà afegir, editar o eliminar un objecte.

A continuació comentarem els detalls de disseny de l'aplicació mòbil.

4.1 Disseny de l'aplicació mòbil

En aquesta secció parlarem del disseny de l'aplicació mòbil; anteriorment hem explicat què volíem fer, ara direm com.

Com s'ha explicat en l'apartat anterior, l'aplicació disposa d'un llistat de socis, blocs i favorits. La idea és que un cop introduïdes i validades les nostres credencials en la pantalla d'identificació es mostrin els tres llistats seguint el disseny: (veure figura 6)

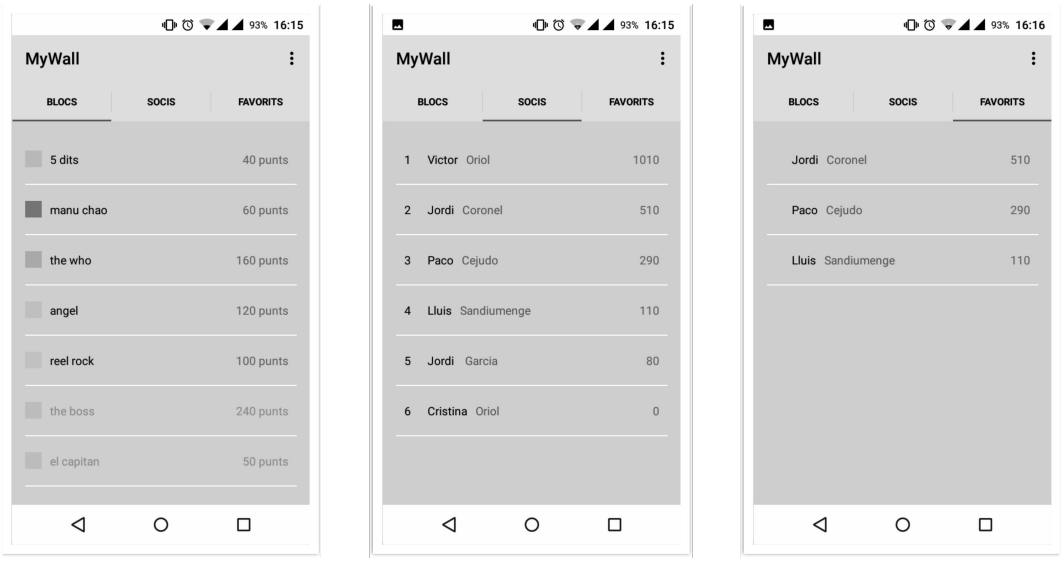

Figura 6: Disseny dels 3 fragments per a l'activity principal de l'aplicació 'MyWall'

El disseny del llistat de blocs s'ha pensat per a que cada fila correspongui a un bloc i mostri la següent informació:

- A l'esquerra, un requadre pintat amb un color, corresponent a la dificultat del bloc[1]. A continuació el nom del bloc i a la dreta la puntuació del bloc[2].

[1]Colors ordenats per dificultat ascendent: rosa, groc, taronja, vermell, marró, verd, blau, lila i negre

[2]Rosa: 40, Groc: 50, Taronja: 60, Vermell: 80, Marró: 100, Verd: 120, Blau: 160, Lila: 200, Negre: 240

Els blocs es poden deshabilitar des de la web d'administració, així quan un bloc es retira de la pared es pot marcar en un to grisenc.

El disseny del llistat de socis s'ha pensat per a que cada fila correspongui a un soci i mostri la següent informació:

- A l'esquerra, un número que indica la posició del soci dintre del rànquing; a continuació, el nom i primer cognom del soci i a la dreta la puntuació total que ha acumulat.

El llistat de favorits segueix la mateixa estructura que el de socis.

Al polsar sobre un bloc trobarem l'opció de sumar aquest bloc a la nostra puntuació, un llistat ordenat per data dels usuaris que també han finalitzat aquesta via i l'opció de veure els comentaris d'altres competidors. Podem veure el disseny en el següent gràfic: (veure figura 7)

Figura 7: Disseny de l'activity d'un bloc on es mostren els socis que han finalitzat aquesta via.

Al polsar el menú d'opcions trobem un botó que ens permet canviar la vista actual a un llistat de comentaris. (veure figura 8)

Figura 8: Disseny de l'activity d'un bloc on es mostren els comentaris.

Al polsar sobre un soci podrem veure els blocs que ha finalitzat i la seva imatge de perfil. (veure figura 9)

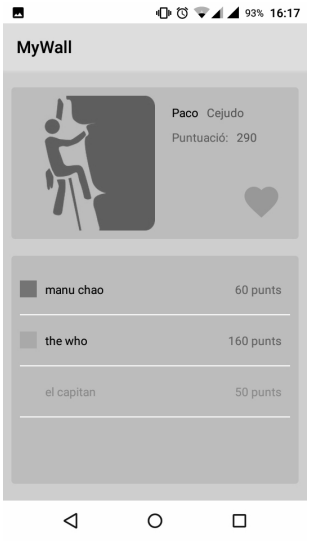

Figura 9: Disseny de l'activity d'un soci.

Un cop explicats i mostrats els dissenys de les activitats principals mostrarem el cicle de vida de l'aplicació: (veure figura 10)

Figura 10: Activity Lifecyrcle de l'aplicació 'MyWall'

Un cop vist el disseny de l'aplicació mòbil passarem a mostrar breument el disseny de la pàgina web d'administració.

4.2 Disseny de l'aplicació web (administració)

La pàgina d'administració és el mètode que utilitzaran els empleats del rocòdrom per gestionar els diferents usuaris i blocs; tots els canvis que s'executin des d'aquest lloc web estaran disponibles a l'aplicació mòbil al moment.

A continuació, mostrarem un gràfic de com quedarà l'aplicació web. (veure figura 11)

Figura 11: Disseny de l'aplicació web d'administració

Un cop definides l'aplicació mòbil i la web, és el moment de parlar del disseny de software del servidor que alimentarà les aplicacions. Ho veiem a continuació.

4.3 Disseny del software Server Side

En el següent gràfic podem veure un petit esquema que mostra l'arquitectura bàsica del projecte. (veure figura 12)

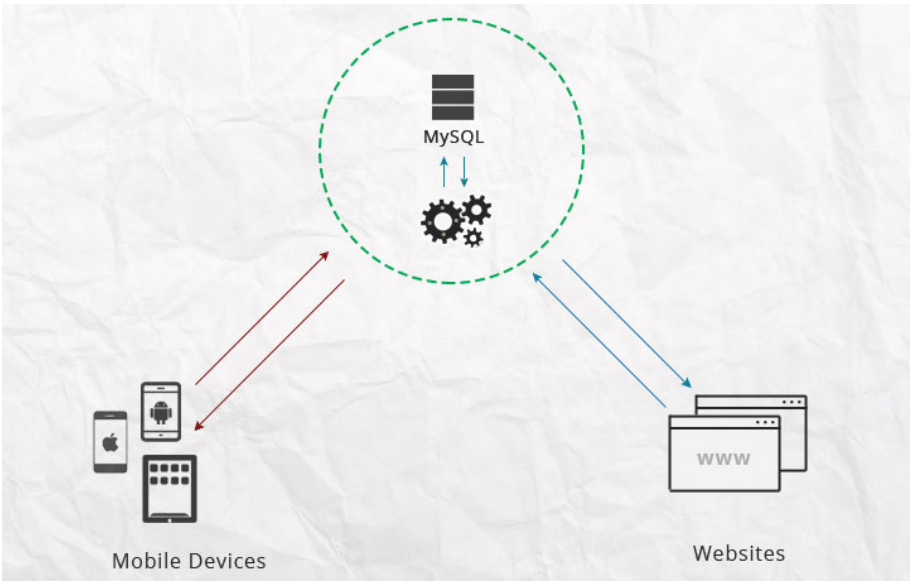

Figura 12: Gràfic amb l'arquitectura software utilitzada

A continuació s'explicarà com s'ha dissenyat, capa a capa, el servidor que alimenta l'aplicació 'MyWall'.

4.3.1 Sistema Operatiu

El sistema operatiu és el conjunt dels diferents programes que controlen el funcionament d'un servidor. Les seves funcions, entre d'altres, consisteixen en gestionar les transferències d'informació internes, procurar la comunicació del servidor amb els operadors, controlar l'execució dels programes amb la detecció dels errors, encadenar automàticament les feines, optimitzar els recursos (memorial, unitat aritmètica, etc.) i carregar i descarregar automàticament els programes en funció de l'espai de memòria i CPU.

S'ha optat per una versió Linux anomenada CentOS; és un clon a nivell binari de la distribució GNU/Linux Red Hat Enterprise Linux (RHEL), compilat per voluntaris a partir del codi font alliberat per Red Hat.
S'ha instal·lat la versió 7, donat que suporta les ultimes versions del programari utilitzat per desenvolupar aquest projecte fent que el nostre sintema sigui mes fiable i durador.

El nostre servidor s'utilitzarà per servir una base de dades i una web. Començarem explicant la capa de dades.

4.3.2 Base de dades

Una base de dades és un conjunt estructurat de dades, organitzades segons una estructura coherent i accessibles des d'un o més programes o aplicacions, de manera que qualsevol d'aquestes dades pot ésser extreta del conjunt i actualitzada, sense afectar ni l'estructura del conjunt ni les altres dades. El programari especialitzat que gestiona aquestes dades s'anomena Sistema Gestor de Bases de Dades.

Per a la realització d'aquest projecte s'ha apostat per Maria DB, una branca del sistema de gestió de bases de dades MySQL impulsada per la comunitat, per tal de mantenir el seu estat lliure sota la GNU GPL.

Es crearà una base de dades MySQL que contindrà els esquemes de Django, dades d'usuari i les dades que se serviran a l'aplicació Android.

Un cop explicada la capa de dades passarem a explicar el servidor web.

4.3.3 Servidor web & RESTful

Un servidor web serveix contingut estàtic a un navegador, carrega un arxiu i el serveix a través de la xarxa al navegador d'un usuari. Aquest intercanvi és intervingut pel navegador i el servidor que parlen l'un amb l'altre mitjançant HTTP.

Es poden utilitzar diverses tecnologies en el servidor per augmentar la seva potència més enllà de la seva capacitat de lliurar pàgines HTML. En aquest cas, s'ha utilitzat el servidor web per lliurar les dades contingudes en la base de dades a l'aplicació mòbil utilitzant RESTful.

La Transferència de Estat Representatiu (Representational State Transfer) o REST és una arquitectura de programari pensada per sistemes distribuïts basats en hipermèdia, com ara el World Wide Web.

Un servei web REST és un model centrat en les dades, en qualsevol format (XML, JSON, etc), sense les abstraccions addicionals dels protocols basats en patrons d'intercanvi de missatges.
Els anomenats recursos vénen identificats per URIs i poden ser manipulats mitjançant accions especificades a les capçaleres HTTP.
Els sistemes que segueixen els principis REST s'anomenen RESTful. (veure figura 13)

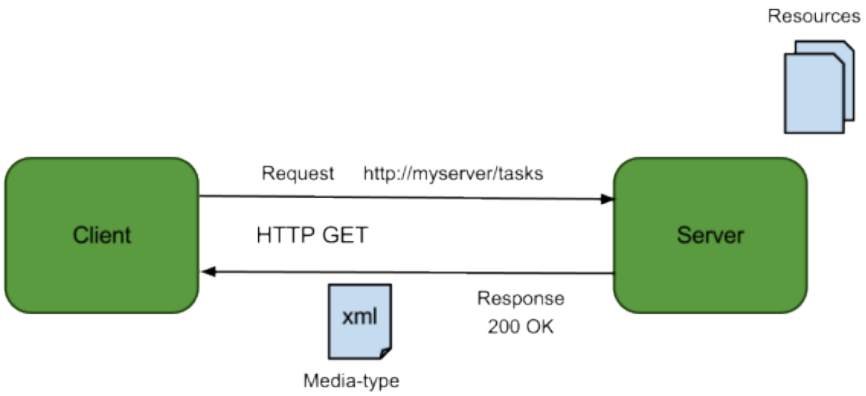

Figura 13: Request de tipus GET mitjançant RESTful Client - Servidor

A continuació es llisten els motius pels quals s'ha decidit utilitzar la tecnologia REST per a l'intercanvi d'informació entre el servidor web i l'aplicació mòbil.

- No necessita tenir un 'estat'.

- Millora el rendiment de l'aplicació.

- Tant el productor com el consumidor coneixen el context i el contingut a intercanviar.

- REST és molt útil pel consum d'un servidor web en dispositius mòbils que disposen de menys recursos.

Donat que aquest projecte és un projecte de concepte, s'ha utilitzat el servidor web de test inclòs en Django Framework per a servir les peticions HTTP.

Amb això tanquem la part d'anàlisi del problema i disseny; però, abans de començar amb el desenvolupament del projecte, explicarem la planificació inicial per dur a terme el projecte 'MyWall'.

4.4 Planificació inicial

A continuació es mostraran unes taules amb les tasques que es creuen necessàries per produir el projecte 'MyWall' i el temps que es calcula que es dedicarà a cadascuna:

Tasca	Temps (h)	Resultat de la tasca
Viabilitat tecnològica	6	Cerca i comparació del software utilitzat.
Viabilitat econòmica	6	Resultat de preus pel desenvolupament del projecte.
Comparativa de mercat	6	Llista de diferents estructures amb el mateix objectiu del projecte per obtenir el millor resultat.

- Preparació de l'entorn de treball

Tasca	Temps (h)	Resultat de la tasca
Adaptació del hardware	12	Adquisició del servidor virtual en el cloud (OVH). Instal·lació del sistema operatiu, actualització del sistema i configuració per poder accedir-hi mitjançant SSH i HTTP .
Instal·lació del software	12	Instal·lació de Maria DB, Django Framework, Django REST Framework i dependències.

- Anàlisi i disseny

Tasca	Temps (h)	Resultat de la tasca
Anàlisi de requeriments d'usuari	18H	Definició dels requeriments per l'usuari final, tant de disseny com de sistema.
Anàlisi de tasques	18	Definició de les tasques que ha d'assolir l'aplicació.
Disseny de software	18	Definició de l'estructura del sistema per aconseguir el seu propòsit.
Prototips i avaluacions - web	12	Resultat del disseny de l'interfície de l'aplicació web.
Prototips i avaluacions - aplicació	18	Resultat del disseny de l'interfície de l'aplicació Android.

- Implementació

Tasca	Temps (h)	Resultat de la tasca
Capa de dades	25	Crear i validar models i inserció de dades de test a la base de dades.
Aplicació web	50	Crear URLs, vistes i 'serialitzar' dades en format JSON.
Aplicació Android	100	Crear l'aplicació, connexió al servidor web i mostrar dades.

- Validació del sistema

Tasca	Temps (h)	Resultat de la tasca
Validació server side	12	Revisar i validar serveis, URLs i la consistència de dades en el costat del servidor.
Validació client side	12	Revisar i validar l'aplicació, excepcions, 'time outs' i la consistència de dades en el costat del client.

- Redacció de la memòria

Tasca	Temps (h)	Resultat de la tasca
Redacció de la memòria	125	Redacció de la memòria

Un cop revisada la planificació inicial del projecte i tenint clar què volem fer és moment de desenvolupar la nostra aplicació. A continuació, expliquem com ho hem fet.

5 Desenvolupament

En aquest apartat s'explicaran totes les tecnologies, configuracions i codi utilitzats per desenvolupar el projecte 'MyWall'. Cada punt té una referència a l'annex on s'explica de forma detallada la implementació duta a terme.
Seguint tots els punts s'obté el projecte sencer; s'han de seguir en ordre donat que en molts casos existeixen dependències.

5.1 Instal·lació i configuració

A continuació, s'explicarà com s'ha instal·lat i configurat l'entorn de treball per poder produir 'MyWall'.

5.1.1 Sistema Operatiu

Com hem comentat anteriorment, s'ha optat per una versió Linux anomenada CentOS. És un clon a nivell binari de la distribució GNU/Linux Red Hat Enterprise Linux (RHEL) compilat per voluntaris a partir del codi font alliberat per Red Hat. S'ha instal·lat la versió 7, donat que suporta les últimes versions del programari utilitzat per desenvolupar aquest projecte fent el nostre sistema més fiable i durador.

Un cop instal·lat i actualitzat el sistema operatiu serà necessari 'securitzar-lo' donat l'elevat nombre d'atacs que es reben a diari mitjançant força bruta sobre aquests sistemes. Així que s'ha configurat SSH per a que no permeti que root accedeixi al sistema directament.

Implementació detallada al punt 8.1.1 del Annex, pàgina 49

Finalitzada la instal·lació, actualització i 'securització' del sistema operatiu ja podem començar a instal·lar els serveis que administrarà. El primer serà un gestor de base de dades on allotjarem totes les dades que servirà l'aplicació.

5.1.2 Base de dades

Per la realització d'aquest projecte s'ha apostat per Maria DB, una branca del sistema de gestió de bases de dades MySQL impulsada per la comunitat, per tal de mantenir el seu estat lliure sota la GNU GPL.

Implementació detallada al punt 8.1.2 del Annex, pàgina 50

Un cop instal·lat el gestor de base de dades necessitem un sistema que ens serveixi les dades en un format que pugui entendre la nostra aplicació mòbil. Per cobrir aquesta necessitat s'ha decidit utilitzar Django Framework. Parlem d'ell a continuació.

5.1.3 Django Framework

Django és un framework per aplicacions web basat en el llenguatge Python.

S'ha utilitzat Django Framework per intercanviar dades entre el servidor web i l'aplicació mòbil.

Implementació detallada al punt 8.1.3 del Annex, pàgina 51

Per facilitar l'intercanvi de dades s'ha decidit utilitzar objectes JSON. El framework de Django REST facilita aquest intercanvi i ofereix eines d'autenticació d'usuaris molt útils. A continuació expliquem com configurar-lo.

5.1.4 Django REST Framework

Django REST Framework és una eina molt potent per desenvolupar APIs Web.

S'ha utilitzat Django REST Framework per convertir els objectes de Django en objectes JSON abans de servir les peticions HTTP amb mètode GET i transformar els objectes JSON obtinguts en peticions HTTP amb mètode PUT en objectes Django. (veure figura 14)
Django REST també inclou un mòdul d'autenticació per a usuaris que s'utilitzarà per a que els socis s'identifiquin a l'aplicació.

Implementació detallada al punt 8.1.4 del Annex, pàgina 51

Per poder servir i rebre dades de l'aplicació mòbil és necessari un servidor web.

5.1.5 Servidor web

Donat que el projecte 'MyWall' és un projecte de concepte, s'ha utilitzat el servidor web de test que inclou Django Framework.

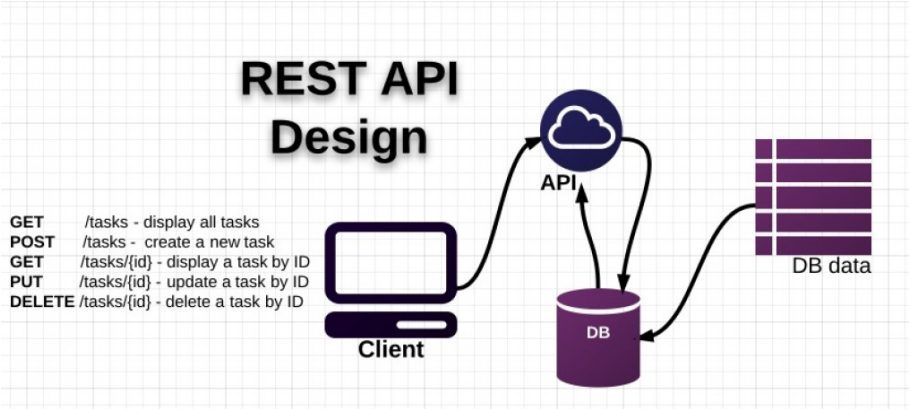

Figura 14: Disseny d'una API RESTful

No es necessari configurar el servidor, només cal executar la comanda següent des de l'arrel del projecte i el servidor web quedarà escoltant peticions en el port 8080:

```
python manage.py runserver 8080
```

El servidor web de desenvolupament de Django és molt lleuger i està escrit purament amb llenguatge Python. Això ens permet desenvolupar funcionalitats ràpidament, sense tenir que configurar un servidor de producció com Apache2.

Per facilitar la gestió del servidor de desenvolupament (stop / start) i poder 'debugar' l'aplicació amb eficiència s'ha convertit la comanda anterior en un servei mitjançant la eina systemctl nativa a Centos 7.
Systemctl és un sistema d'arranc i gestor de serveis que està reemplaçant al sistema tradicional SysV a Linux.

Implementació detallada al punt 8.1.5 del Annex, pàgina 52

Un cop hem vist com instal·lar i programar l'entorn que produirà les dades que alimentaran l'aplicació mòbil, parlarem de com implementar les diferents funcionalitats del projecte.

5.2 Implementació

En aquest apartat parlarem de com s'han implementat les diferents funcionalitats que han donat lloc al projecte 'MyWall'. Com en l'apartat anterior, en cada punt s'ha fet referència a l'annex on s'explica de forma detallada l'implementació duta a terme.

5.2.1 Nou projecte i aplicació de Django

El pseudocodi per a crear un nou projecte i aplicació a Django es el següent:

1.- Es crea un nou projecte de base.

2.- Es crea una nova base de dades i es relaciona amb el projecte creat anteriorment.

3.- Després es crea una una nova aplicació dintre del projecte.

Implementació detallada al punt 8.1.6 del Annex, pàgina 52

Un cop creada la nova aplicació a Django, el següent pas es definir el model de dades.

5.2.2 Estructura de Django

Un nou projecte de Django té la següent estructura:

```
djangoDD/
    manage.py
        mysite/
        __init__.py
        settings.py
        urls.py
        wsgi.py
```

A continuació explicarem quin es el propòsit de cada fitxer:

- manage.py: Aquest fitxer conté el conjunt de funcions Python que ens permet administrar el nostre projecte i aplicacions de Django, com ara: sincronitzar els models amb la base de dades, agrupar els fitxers estàtics en el directori arrel, engegar el servidor de test, etc.

- __init__.py: És un fitxer generat per Python per indicar que el directori que el conte és un Python Package).

- settings.py: És el fitxer de configuració del projecte.

- urls.py: És el fitxer on es defineixen les URIs globals, és a dir, no té a veure amb les URL de l'aplicació.

- wsgi.py: Aquest fitxer permet integrar el projecte amb un servidor web.

A l'estructura base del projecte s'han d'afegir manualment els directoris media/-profile. És on es desaran les imatges de perfil dels socis.

L'estructura per defecte de l'aplicació és la següent:

```
ranking/
    __init__.py
    admin.py
    models.py
    tests.py
    views.py
```

On el propòsit de cada fitxer és:

- admin.py: És el fitxer que enllaça els models a l'administració de Django.

- models.py: És el fitxer on es defineix la classe de cada objecte, la nomenclatura a l'hora de definir les dades és molt semblant a MySQL.

- test.py: És el fitxer on es configuren els tests de l'aplicació.

- view.py: És el fitxer on es defineixen les vistes per cada URL de l'aplicació.

A l'estructura base de l'aplicació hem d'afegir els següents fitxers:

- urls.py: Aquest fitxer ens permet definir les URI de l'aplicació.

- serializers.py: Aquest fitxer contindrà les funcions que convertiran els objectes de Django en JSONs.

Un cop tenim clara l'estructura d'un projecte i aplicació de Django el següent pas és definir el model de dades.

5.2.3 Model de dades de Django

Un cop creat el projecte i la nova aplicació és necessari definir l'estructura o model de dades. A continuació mostrarem el diagrama de dades dissenyat per a l'aplicació 'ranking' de Django (veure figura 15)

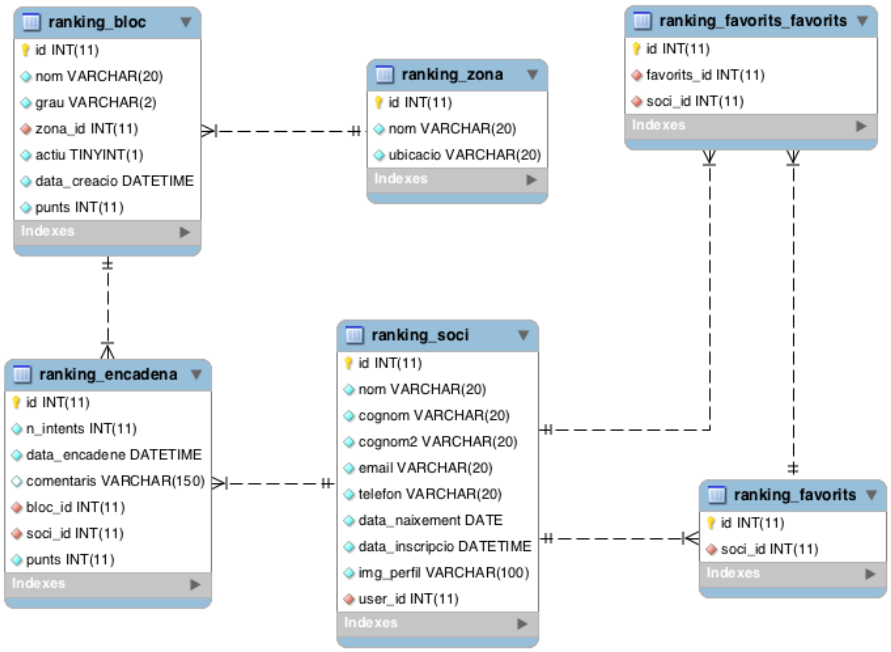

Figura 15: Diagrama de dades de l'aplicació 'MyWall'

Després de definir el model en el fitxer models.py de l'aplicació és necessari sincronitzar Django amb la base de dades.

Implementació detallada al punt 8.1.7 del Annex, pàgina 53

Un cop hem definit el model de dades i hem sincronitzat l'aplicació amb la base de dades és el moment de definir les URIs de l'aplicació.

5.2.4 Definició de URLs de Django

Django permet definir nombroses URLs mitjançant expressions regulars (regex), caràcters que tenen un significat especial, anomenats meta-caràcters.
Cada URI es relaciona a una vista mitjançant el fitxer urls.py (/djangoDD/ranking/urls.py) de l'aplicació (veure figura 16)

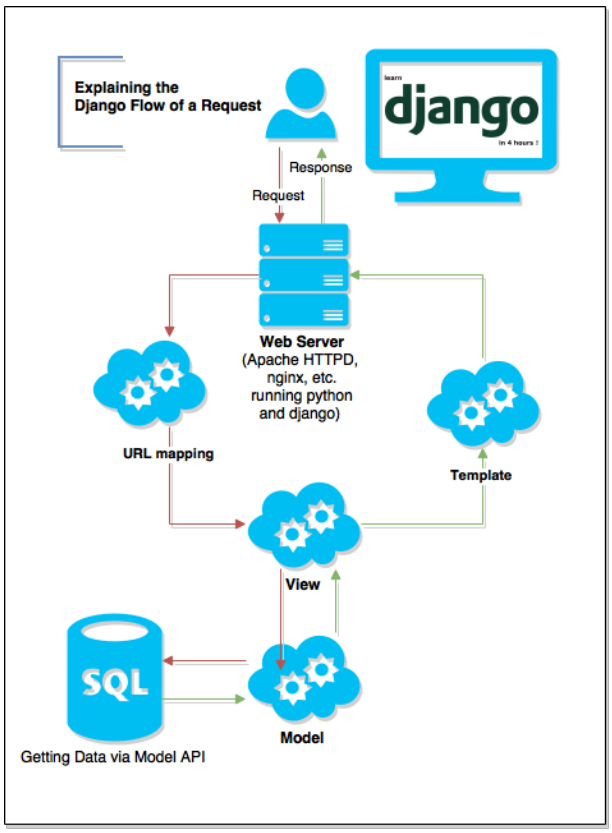

Figura 16: Diagrama que mostra el flux d'una request a Django

Implementació detallada al punt 8.1.8 del Annex, pàgina 55

Un cop configurades les URI hem de definir les vistes associades. Ho veurem en el següent apartat.

5.2.5 Vistes de Django

Una vista és un tipus de pàgina web dintre de l'aplicació de Django, que generalment serveix per a una funció especifica i té relacionada una 'template' especifica. En aquest projecte no se serveixen pàgines HTML o PHP com a tal, si no objectes JSON.
Cada funció 'serialitzarà' els objectes filtrats en el model i els retornarà en format JSON a les crides HTTP amb el mètode GET o executarà operacions CRUD en la base de dades a partir d'un objecte JSON per a les crides HTTP amb el mètode PUT.

Així doncs es crearà una vista per a cada URI definida en el fitxer urls.py (/djangoDD/ranking/urls.py) de l'aplicació.

Implementació detallada al punt 8.1.9 del Annex, pàgina 56

Per convertir els objectes del model en JSON i viceversa s'ha utilitzat Django REST. Expliquem com funciona a continuació.

5.2.6 'Serialitzant' objectes de Django

Django REST Framework inclou un mecanisme per 'traslladar' models de Django a altres formats. Per poder enviar i rebre informació de forma senzilla a l'aplicació 'MyWall' s'han 'traslladat' aquestes dades a objectes JSON.

En la secció de 'Instal·lació i configuració' hem explicat com instal·lar Django REST Framework, podem trobar la informació detallada al punt 8.1.4 del Annex, pàgina 51.
Un cop instal·lat el framework només és necessari afegir el fitxer serializers.py a l'arrel de l'aplicació Django i definir les funcions que indiquin quines dades es volen incloure en l'objecte JSON que s'afegirà a la HTTP response.

Implementació detallada al punt 8.1.10 del Annex, pàgina 61

Un cop podem rebre i retornar objectes JSON veurem com configurar Django per a poder desar fitxers estàtics.

5.2.7 Fitxers estàtics a Django

S'entén com a fitxers estàtics aquells utilitzats per l'aplicació web i que no es modifiquen després de la seva creació.

El projecte té configurat un directori específic per contenir tots els estàtics i és imperatiu que tots estiguin allà per a que el servidor web hi pugui accedir.

S'han utilitzat els fitxers estàtics per desar les imatges de perfil dels socis.

Implementació detallada al punt 8.1.12 del Annex, pàgina 65

Fins ara hem explicat com instal·lar i configurar Django per retornar i rebre peticions HTTP sense restriccions, però per un tema de confiança no volem que qualsevol tingui accés a les nostres dades, així que s'ha configurat 'TokenAuthentication' (inclòs en el paquet de Django REST). Ho veiem a continuació.

5.2.8 'TokenAuthentication' de Django REST

L'esquema d'autenticació 'TokenAuthentication' de Django REST Framework usa un esquema d'autenticació HTTP basat en tokens.
S'ha utilitzat aquest sistema per dos motius: el primer és que aquest sistema d'autenticació és molt apropiat per a configuracions client-servidor en aplicacions mòbils. La segona és que l'autenticació 'TokenAuthentication' resulta suficient per a un projecte de concepte. (veure figura 17)

L'autenticació de Django REST utilitza la gestió d'usuaris de Django, per tant hem de relacionar el model Soci de l'aplicació 'ranking' de Django amb els users de Django per a que els socis utilitzin aquest mètode d'autenticació.

Afegirem la línia següent al model Soci del fitxer models.py (djangoDD/ranking/-models.py) de l'aplicació Django.

```
class Soci(models.Model):
        user = models.ForeignKey('auth.User', related_name='s_user')
```

D'aquesta manera els usuaris de Django i els socis de l'aplicació queden relacionats i l'autenticació es delega completament a Django.
En cas de no estar correctament autenticats es rebrà un codi d'error HTTP 401 (Unauthorized) en lloc del recurs sol·licitat en cada petició.

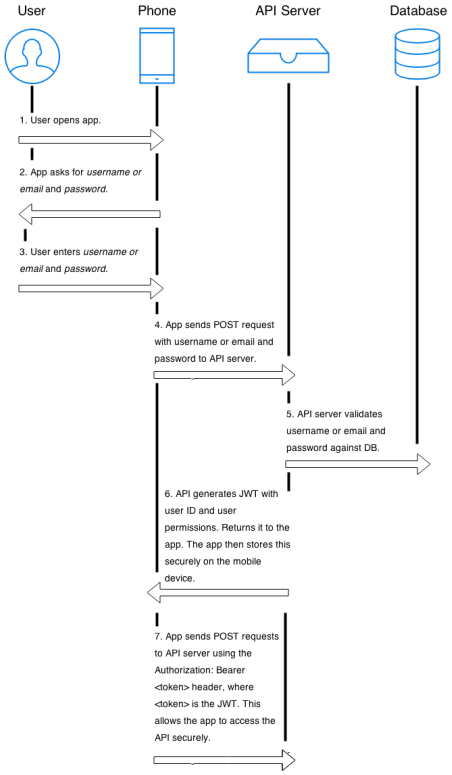

Figura 17: Diagrama que mostra el flux d'una request a Django

Implementació detallada al punt 8.1.11 del Annex, pàgina 64

Un cop tenim a Django donant servei, explicarem com s'ha configurat la pàgina d'administració per a que els encarregats del rocòdrom pugin gestionar els diferents socis i blocs de la sala.

5.2.9 Pàgina d'administració a Django

En aquest últim punt sobre la implantació de Django en el projecte 'MyWall' veurem com configurar la pàgina d'administració.
La pàgina d'administració és el mètode que utilitzaran els empleats del rocòdrom per a gestionar els diferents usuaris i blocs, tots els canvis que s'executin des d'aquest lloc web estaran disponibles a l'aplicació mòbil al moment.

Per defecte, Django disposa d'una web d'administració per a l'usuari admin (super user). Per accedir només és necessari afegeix 'admin/' a la URL del projecte. Aquesta pàgina requereix d'autenticació.
Un cop tenim els models definits en el fitxer de models.py de l'aplicació és possible crear rols d'usuari i grups donant permisos per crear, modificar i eliminar objectes del model des de la web d'administració de forma gràfica . (veure figura 18)

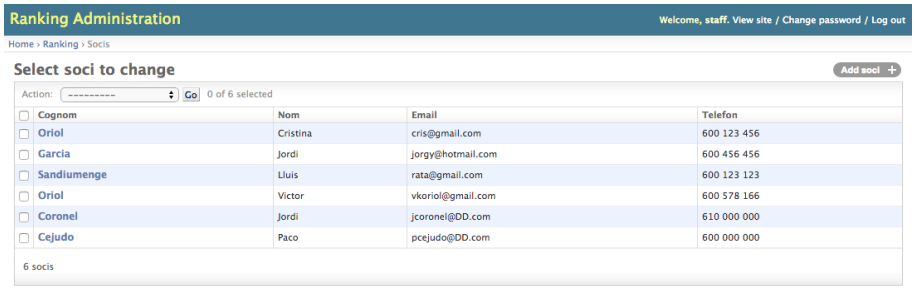

Figura 18: Pàgina de i per a l'administració de blocs, socis i zones per part dels empleats.

Implementació detallada al punt 8.1.13 del Annex, pàgina 66

Amb aquest punt finalitza tota la documentació relacionada amb la instal·lació, configuració, implementació i ús de Django. En endavant es parlarà sobre l'aplicació mòbil o producte final.

5.2.10 Nou projecte a Android Studio

A continuació veurem pas a pas com s'ha construït l'aplicació d'Android 'MyWall'. S'ha utilitzat Android Studio, un entorn de desenvolupament per a la plataforma Android.

L'instal·lació d'Android Studio és senzilla (següent, següent, següent...), el programari es pot descarregar fàcilment des de la pagina oficial de l'aplicació `http://developer.android.com/sdk/index.html`.

Un cop instal·lat l'entorn, crearem un nou projecte indicant el nom de l'aplicació i el 'Minimum SDK'[3]. En el nostre cas, hem escollit la API 15, la recomanada per Google.
A continuació, l'entorn ens preguntarà si volem partir d'alguna 'activity'[4] 'preconfigurada'. Com el primer que farà un soci en engegar l'aplicació és identificar-se hem escollit començar amb una 'Login Activity'. Definim un nom i ja hem acabat! Ja podem començar a programar la lògica de l'aplicació.

[3]Versió mínima de sistema operatiu Android que demanarà l'aplicació per funcionar

[4]Una 'activity' és la pantalla sobre la que treballarem, el projecte disposarà de diverses 'activities'

5.2.11 Identificar-se a l'aplicació 'MyWall'

L'activitat de 'login' (LoginActivity.java) disposa del seu propi 'layout' (activity_login.xml). Un 'layout' fa la funció de 'view' (MVC) dintre d'una aplicació d'Android; està escrit en llenguatge XML i si és necessari, cada component es pot cridar des del codi per modificar-lo. En aquesta 'activity' no ens hem de preocupar del disseny, donat que parteix d'una 'template', però sí de modificar els strings que es mostren, donat que l'aplicació 'MyWall' s'ha dissenyat en català. (veure figura 19)

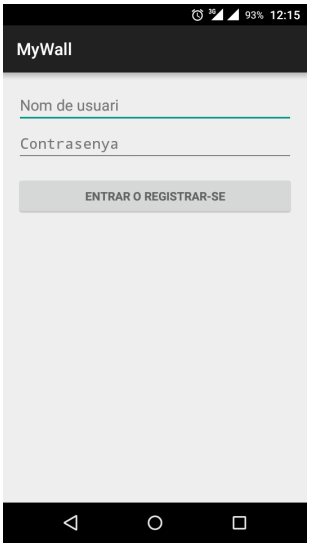

Figura 19: Login Activity de l'aplicació 'MyWall'

El primer que farà l'activitat és validar si ens hem identificat anteriorment. En cas afirmatiu, ens enviarà a l'activitat principal de l'aplicació (MainActivity.java), que és on es mostren el llistat de blocs, socis i favorits.
En cas de no estar identificats carregarà el 'layout' del 'login' i en polsar el botó de identificar-se el codi llançarà una AsyncTask que enviarà el nostre usuari i contrasenya al servidor per validar que les nostres credencials són correctes (LoginService.java).

Si les nostres credencials són correctes el servidor ens retornarà el nostre 'Token' personal per a que l'afegim a la resta de peticions. En cas contrari, ens retornarà un error 401.

Si el login és correcte es llançarà l'activitat principal de l'aplicació, el llistat de blocs, socis i favorits. A continuació veurem com funciona.

5.2.12 Activitat principal de l'aplicació 'MyWall'

Un cop superada la identificació de usuari es carrega l'activitat principal de l'aplicació (MainActivity.java). S'ha configurat aquesta 'activity' per a que hereti de la classe FragmentActivity i poder navegar a través dels llistats de blocs, socis i favorits arrossegant horitzontalment la pantalla.

En utilitzar FragmentActivity es permet intercanviar diferents Fragments[5] lliscant la pantalla horitzontalment. (veure figura 20)

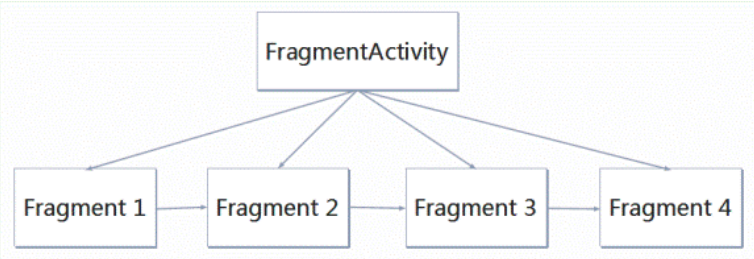

Figura 20: Senzill gràfic que mostra el funcionament del FragmentActivity

Cada llistat (blocs, socis i favolits) està lligat a un Fragment (BlocFragment.java, SociFragment.java, FavoritsFragment.java) i té el seu propi disseny (fragment_blocs.xml, fragment_socis.xml i fragment_favorits.xml) (veure figura 21)

A continuació explicarem detalladament com s'ha aconseguit el llistat de blocs a partir del fragment BlocFragment. El llistat de socis i favorits segueixen la mateixa metodologia.

S'ha programat una petita llibreria (AndroidNetworkUtility.java) que ens permet saber si el nostre dispositiu està connectat a la xarxa i analitzar les 'HTTP Responses'.
Doncs bé, el primer que farem és validar que estem connectats a la xarxa; en cas afirmatiu, llançarem una AsyncTask (BlocAsyncTask.java) per a obtenir el llistat de blocs del nostre servidor web.

[5]Un Fragment representa un comportament o una porció de la interfície d'usuari d'una activitat Android

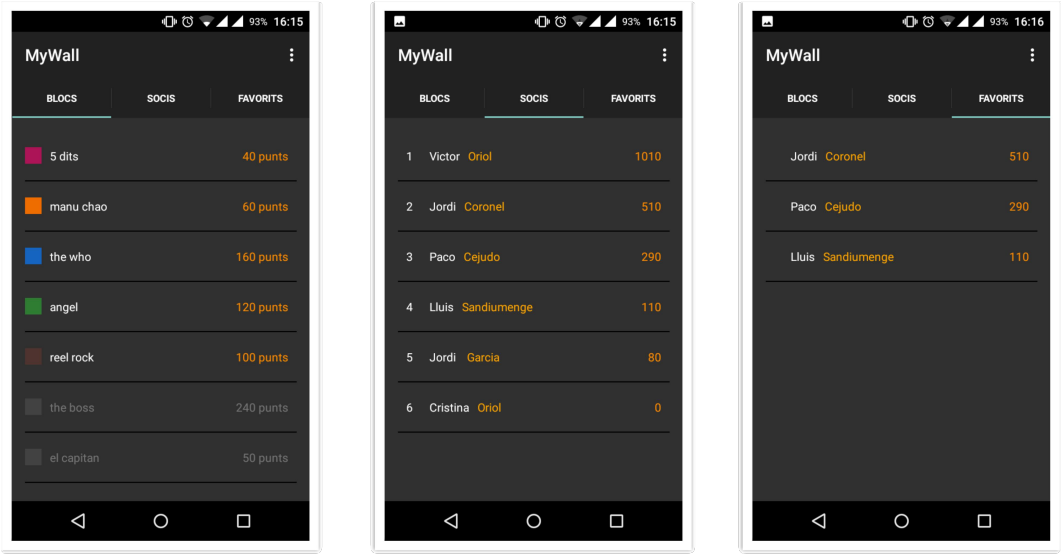

Figura 21: Disseny dels tres fragments que componen l'activitat principal de l'aplicació 'MyWall'

Per què una AsyncTask? Perquè per a utilitzar els mètodes de Apache HttpGet i HttpPost és necessari fer-ho en segon pla. Per aquest motiu s'ha utilitzat el mètode 'doInBackground' de la classe AsyncTask per a enviar i rebre JSONs al servidor web.
Aprofitant la classe, s'ha utilitzat la funció 'onPostExecute' per incloure una 'progress bar' que dóna un efecte de "estem treballant per descarregar les dades"a l'usuari. Sense aquest detall, quan la senyal de xarxa és dèbil fa l'efecte que l'aplicació no funciona o no carrega dades.

El mètode 'doInBackground' comentat anteriorment cridarà a la classe BlocService (BlocService.java) que farà una petició httpget al nostre servidor web.
La petició està composta per la URL del recurs que volem rebre i una capçalera amb el 'Token' del usuari.

Si la resposta és correcta es rebrà un JSON amb el llistat de blocs que posteriorment es convertirà a un ArrayList de blocs.
Un cop disposem del llistat de blocs a l'aplicació s'inflaran les dades mitjançant un ArrayAdapter (BlocArrayAdapter.java).

A l'hora d'afegir els diferents blocs al 'Layout' (listview_row_layout_blocs.xml) s'afegirà un 'listener'[6] a cada bloc per a que al polsar-lo es pugui llançar una nova activitat.
El resultat d'aquest procés és el Fragment finalitzat. (veure figura 22)

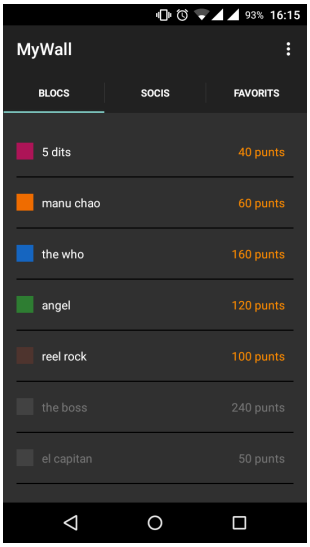

Figura 22: Disseny del llistat de blocs (BlocFragment)

A continuació veurem com s'han produït les activitats per mostrar un bloc o soci.

5.2.13 Activitats secundàries de l'aplicació 'MyWall'

Des de l'activitat principal es pot accedir a un bloc o soci polsant sobre la fila d'un dels dos llistats. Això és possible gràcies al 'listener' que s'associa a cada bloc i soci quan es carrega l'activitat principal.

Començarem parlant de l'activitat d'un bloc.
En carregar un bloc es llança una AsyncTask (utilitzant el mètode explicat a la secció anterior) per obtenir els socis que han encadenat el bloc, aquest llistat manté el format del llistat de l'activitat principal.
Si el bloc que ha carregat un soci no s'ha encadenat és possible fer-ho polsant el botó 'ENCADENA!' (veure figura 23)

[6]Els objectes declarats com a oïdors ('listeners') del control reben l'esdeveniment i, immediatament, executen algun dels mètodes associats a la seva condició de 'listeners'.

Figura 23: Disseny de l'activitat d'un bloc (BlocOnClick).

En polsar el botó 'ENCADENA!' s'executarà una classe tipus AsyncTask que enviara un HTTP POST al servidor web amb les dades del soci i el bloc a encadenar; aquest post iniciarà un procés en el costat del servidor que conclourà amb un insert a la base de dades.

En polsar el botó d'opcions de l'activitat podrem canviar la vista de forma que en lloc de mostrar el llistat de socis que han encadenat un bloc mostrarà els seus comentaris. (veure figura 24)

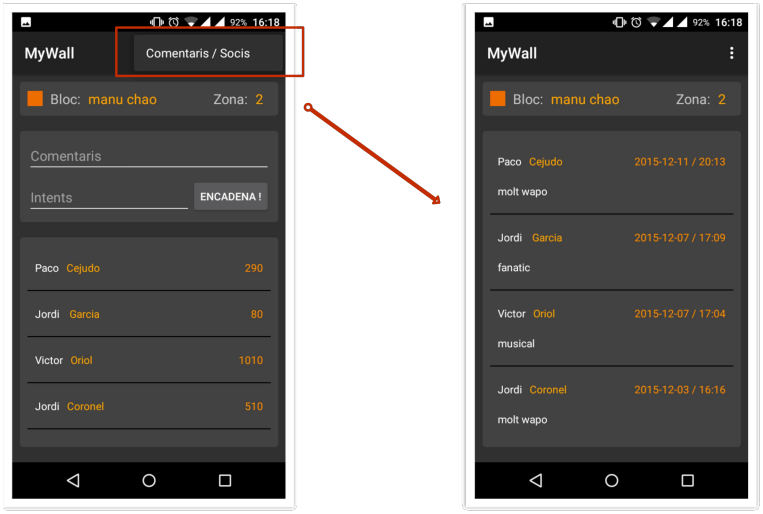

Figura 24: Figura que mostra les dues possibles vistes d'un bloc (BlocOnClick).

Un cop explicada l'activitat bloc, continuarem amb la de soci.
En carregar un soci es llança una AsyncTask (utilitzant el mètode explicat a la secció anterior) per obtenir els blocs que ha encadenat el soci. Com en el cas dels blocs, aquest llistat manté el format del llistat de l'activitat principal.
És possible afegir a un soci al nostre llistat de favorits polsant sobre la icona amb forma de cor i de la mateixa manera es pot eliminar. (veure figura 25)

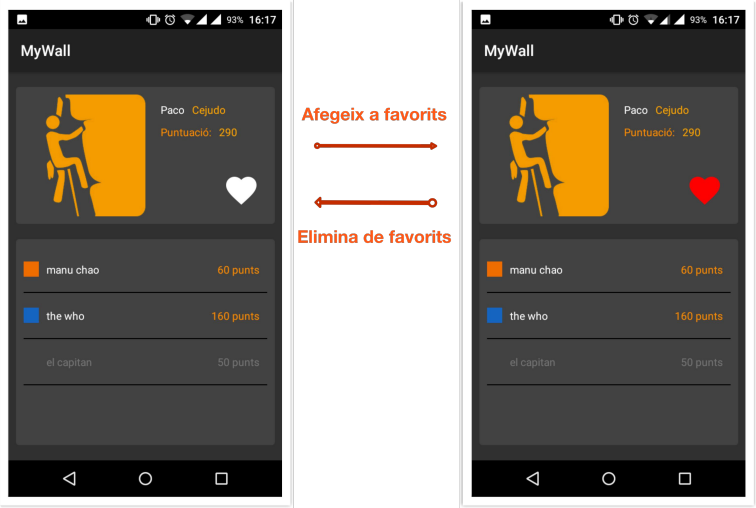

Figura 25: Disseny de l'activitat d'un soci (SociOnClick) desmarcat (esquerra) i marcat (dreta) com a favorit

Si el soci no es troba a la nostra llista de favorits, en polsar la icona s'executarà una classe tipus AsyncTask que enviara un HTTP POST al servidor web amb les dades del soci a afegir al nostre llistat de favorits. Aquest post iniciarà un procés en el costat del servidor que conclourà amb un insert o 'update' a la base de dades. Si volem treure de la nostra llista de favorits el soci només és necessari tornar a polsar la icona i s'executarà una classe tipus AsyncTask que enviarà un HTTP POST al servidor web amb les dades del soci a afegir al nostre llistat de favorits. Aquest post iniciarà un procés en el costat del servidor que conclourà amb un 'delete' a la base de dades.

Un cop hem parlat de les activitats secundàries de l'aplicació explicarem un mètode diferent per accedir a un bloc mitjançant la lectura d'un codi QR.

5.2.14 Carregar un bloc des d'un codi QR a l'aplicació 'MyWall'

Per a potenciar l'aplicació i fer-la mes dinàmica s'ha decidit codificar les URL associades a cada bloc amb un codi QR per poder accedir a un determinat bloc des de la càmera del nostre telèfon mòbil. (veure figura 26)

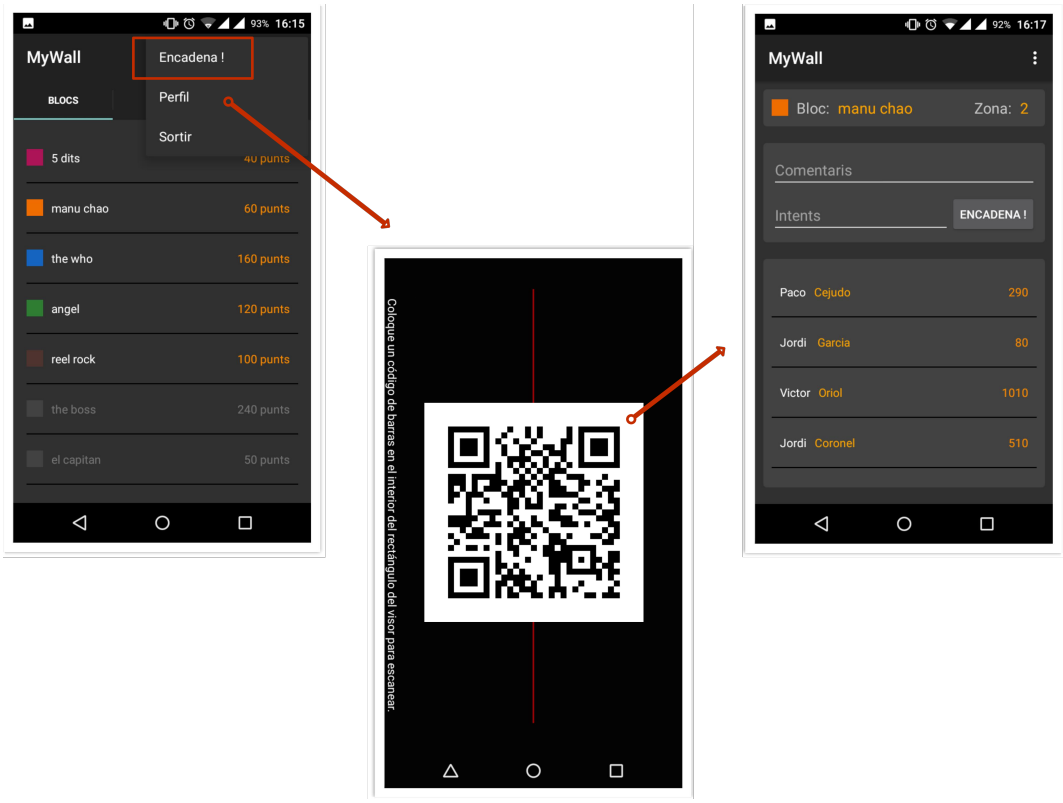

Figura 26: Gràfic que mostra com accedir a un bloc amb el lector de codis QR

Per implementar l'accés a un bloc a partir de la lectura d'un codi QR s'ha utilitzat la llibreria de google zxing. Per utilitzar aquesta llibreria s'han de fer els imports adients en la classe des d'on es cridarà:

```
import com.google.zxing.integration.android.IntentIntegrator;
import com.google.zxing.integration.android.IntentResult;
```

Per poder utilitzar la classe IntentIntegrator i IntentResult. El pseudocodi és el seguent:

- La classe IntentIntegrator ens permetrà llançar un nou intent per capturar el codi QR en polsar el botó de les opcions 'Encadena!'

- Un cop capturat el codi, la classe IntentResult ens permetrà convertir-lo en un String amb la URL del bloc.

- Si la URL és vàlida s'executarà una AsyncTask per obtenir la informació del bloc del servidor i posteriorment llançar l'activitat 'BlocOnClick'.

Un cop explicat com accedir a un bloc des d'un codi QR, explicarem com podem modificar algunes de les dades del nostre usuari des de l'aplicació mòbil.

5.2.15 Modificar perfil a l'aplicació 'MyWall'

Des del menú d'opcions de l'activitat principal es permet accedir a la configuració del perfil, que compta amb dues possibles opcions de configuració: modificar la contrasenya de l'usuari o la imatge de perfil. (veure figura 27)

Ambdues opcions llancen una classe AsyncTask amb un mètode que envia un HTTP POST al servidor per actualitzar la contrasenya de l'usuari o la imatge de perfil.

Amb tot allò comentat amb anterioritat s'ha explicat com desenvolupar el projecte 'MyWall', tant la part del servidor com l'aplicació mòbil. El següent pas serà analitzar els resultats obtinguts. Ho veiem en la següent secció.

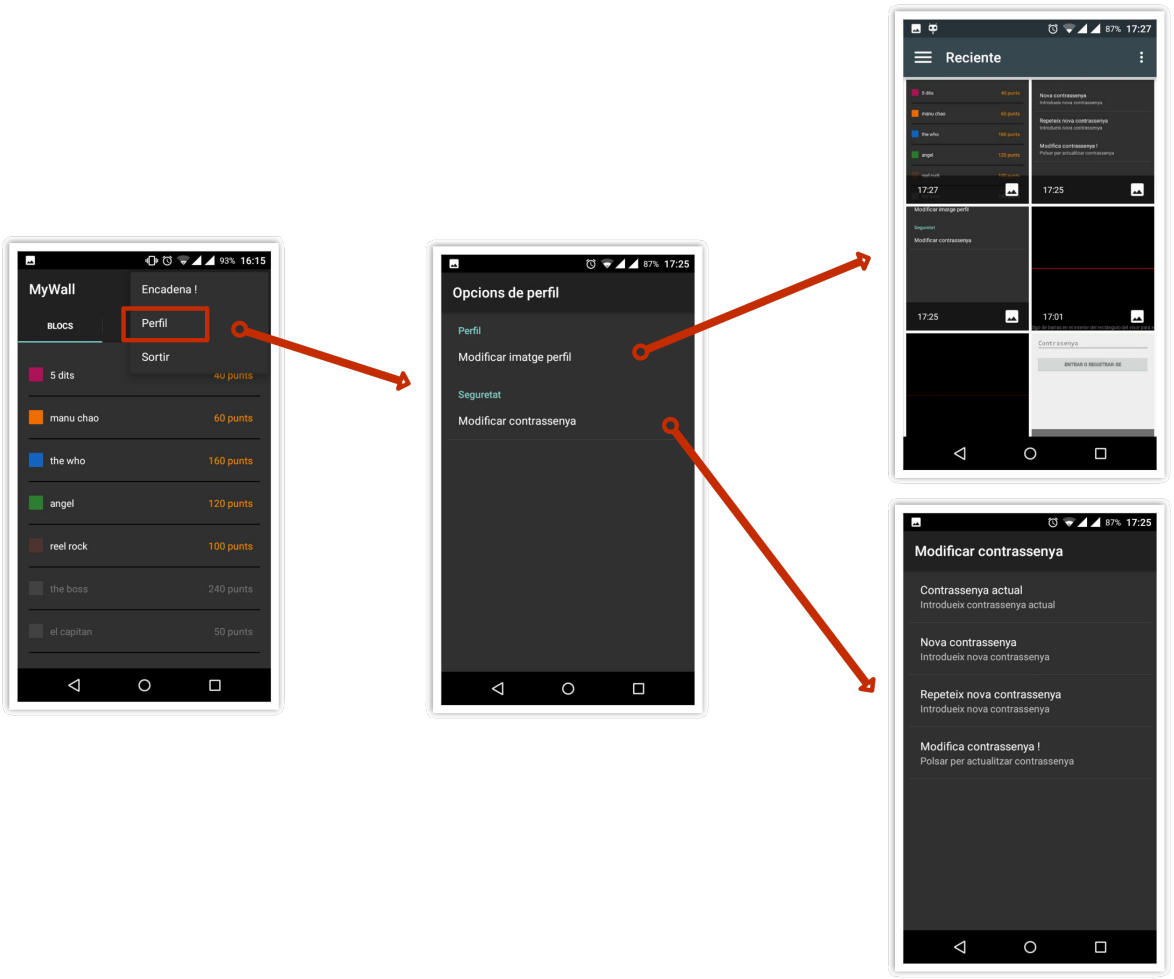

Figura 27: Gràfic que mostra el flux de les opcions del perfil.

6 Resultats

A continuació, parlarem dels resultats obtinguts. Els dividirem en dues seccions, l'aplicació mòbil per als usuaris i la web d'administració.

6.1 Resultats obtinguts de l'aplicació mòbil

En la secció d'implementació s'han anat mostrant diverses captures amb el disseny final de l'aplicació i el funcionament de la mateixa, amb l'objectiu d'entendre millor les diferents tecnologies utilitzades i implementades. Per aquest motiu en aquesta secció farem un anàlisi més explicatiu que no gràfic.

El resultat de l'aplicació 'MyWall' compleix perfectament els objectius plantejats al principi del projecte, donant lloc a una aplicació dinàmica i fàcil d'utilitzar. Com s'ha comentat en la secció d'anàlisi, és un requisit indispensable disposar de connexió a Internet per a utilitzar l'aplicació, ja que tots els continguts dinàmics s'obtenen del servidor web.
És per això que la velocitat de l'aplicació ve definida per la cobertura de xarxa que tinguem en aquell moment. S'han realitzat tests de connectivitat amb xarxes 3G, H+ i Wifi i tots els resultats han tingut èxit. No obstant això, s'ha trobat un retard de diversos segons en carregar el llistat de blocs quan la qualitat de la xarxa es baixa.

A continuació parlarem de l'aplicació web o portal d'administració.

6.2 Resultats obtinguts de l'aplicació web

L'aplicació web està dissenyada per utilitzar-se des d'un ordinador de sobretaula, un ordinador portàtil o una tauleta tàctil. En tots els casos els resultats han sigut els esperats. Una aplicació senzilla però robusta, basada en llistats i botons per afegir, modificar o eliminar objectes amb un sol click.

En executar qualsevol inserció, modificació o eliminació en la web d'administració té un efecte immediat en l'aplicació mòbil, donat que afecta directament a la base de dades.

En el següent punt parlarem de l'experiència dels usuaris al provar l'aplicació 'MyWall' en un telèfon intel·ligent de baix cost (Motorola Moto E).

6.3 Experiència dels usuaris

Un cop finalitzat el prototip de l'aplicació 'MyWall' s'ha permès a alguns socis del gimnàs DeuDits, ubicat a la ciutat de Barcelona, provar l'aplicació amb les seves pròpies mans i l'han trobat una aplicació molt intuïtiva i fàcil de fer servir, útil i amb futur dintre del gimnàs.

Els socis més exigents han trobat a faltar la socialització de l'aplicació mitjançant xarxes socials. S'ha proposat com a possible millora dintre de la secció 'Treball futur' que veurem més endavant.

A continuació i un cop finalitzat el projecte, es mostrarà el temps real empleat en cada tasca del projecte 'MyWall'.

6.4 Planificació final

Un cop finalitzat el projecte s'han revisat els càlculs fets en la planificació inicial durant l'estudi de viabilitat i s'han trobat algunes diferències entre el que es va planificar i el còmput real. A continuació, es mostra el temps aproximat dedicat a cada tasca duta a terme en el projecte:

Tasca	Temps (h)	Resultat de la tasca
Viabilitat tecnològica	8	Cerca i comparació del software utilitzat.
Viabilitat econòmica	2	Resultat de preus pel desenvolupament del projecte.
Comparativa de mercat	8	Llista de diferents estructures amb el mateix objectiu del projecte per obtenir el millor resultat.

- Preparació de l'entorn de treball

Tasca	Temps (h)	Resultat de la tasca
Adaptació del hardware	8	Adquisició del servidor virtual en el cloud (OVH). Instal·lació del sistema operatiu, actualització del sistema i configuració per poder accedir-hi mitjançant SSH i HTTP .
Instal·lació del software	8	Instal·lació de Maria DB, Django Framework, Django REST Framework i dependències.

- Anàlisi i disseny

Tasca	Temps (h)	Resultat de la tasca
Anàlisi de requeriments d'usuari	25	Definició dels requeriments per l'usuari final, tant de disseny com de sistema.
Anàlisi de tasques	12	Definició de les tasques que ha d'assolir l'aplicació.
Disseny de software	12	Definició de l'estructura del sistema per aconseguir el seu propòsit.
Prototips i avaluacions - web	10	Resultat del disseny de l'interfície de l'aplicació web.
Prototips i avaluacions - aplicació	8	Resultat del disseny de l'interfície de l'aplicació Android.

- Implementació

Tasca	Temps (h)	Resultat de la tasca
Capa de dades	25	Crear i validar models i inserció de dades de test a la base de dades.
Aplicació web	75	Crear URLs, vistes i 'serialitzar' dades en format JSON.
Aplicació Android	125	Crear l'aplicació, connexió al servidor web i mostrar dades.

- Validació del sistema

Tasca	Temps (h)	Resultat de la tasca
Validació server side	8	Revisar i validar serveis, URLs i la consistència de dades en el costat del servidor.
Validació client side	16	Revisar i validar l'aplicació, excepcions, 'time outs' i la consistència de dades en el costat del client.

- Redacció de la memòria

Tasca	Temps (h)	Resultat de la tasca
Redacció de la memòria	100	Redacció de la memòria

Un cop comentats els resultats finals del projecte i revisat el temps real dedicat a cada tasca, parlarem de les conclusions obtingudes i plantejarem millores de cara a un treball futur.

7 Conclusions i Treball futur

L'objectiu principal del projecte era construir una lliga entre els socis d'un rocòdrom. Aquest objectiu s'ha assolit, donat que cada soci pot anotar-se una via que hagi finalitzat i obtenir la puntuació corresponent en funció de la dificultat i el nombre d'intents que hagi necessitat per a completar el bloc.

La lliga resultant està reglada pels administradors del gimnàs i actualitzada des del portal web d'administració. Crear una lliga estable formava part dels objectius del projecte, per tant aquest problema també ha quedat solucionat.

A més a més, la possibilitat de generar una petita lliga afegint favorits a la nostra llista formava part dels objectius secundaris i s'ha integrat tal com s'esperava. Donant lloc a una aplicació més propera i customitzable.

En acabat, comentar que l'experiència de desenvolupar el projecte ha estat satisfactòria i molt gratificant donat que, com s'ha comentat a l'inici de la memòria, tant l'àmbit del projecte com les eines utilitzades per dur-lo a terme formen part del meu dia a dia.
Durant la realització d'aquest projecte he pogut consolidar els diferents coneixements obtinguts durant els estudis de grau d'informàtica i demostrar que sóc capaç de tirar endavant un projecte de gran envergadura amb esforç i dedicació.

A continuació, parlarem de quines funcionalitats es podrien afegir al projecte en cas de voler continuar.

7.1 Treball futur

En cas de voler continuar treballant en aquest projecte, es podrien implementar tecnologies de la informació per fer més potent i comercial l'aplicació 'MyWall', com són la 'gamification' i la 'socialization'.

- Gamification: La ludificació o gamificació és l'ús dels elements i de la mecànica del joc en contextos aliens a aquest, amb l'objectiu d'orientar el comportament de les persones i aconseguir determinades fites.
Fent visibles recompenses aconseguides en assolir determinats objectius augmentaria la competició entre els socis i l'ús de l'aplicació. (veure figura 28)

Figura 28: Imatge que mostra gràficament el concepte de gamificació.

- Socialization: Integrar les xarxes socials a la nostra aplicació i donar l'opció a compartir els assoliments obtinguts durant una sessió d'escalada motivarà l'enregistrament de blocs finalitzats, facilitarà la difusió de l'aplicació i incentivarà a nous seguidors. (veure figura 29)

Figura 29: Imatge que mostra 5 de les xarxes socials més utilitzades a Espanya.

Amb aquesta secció i el que hem vist anteriorment es disposa de la informació necessària per entendre, dur a terme i continuar aquest projecte. Per acabar, a la biografia es podran veure els llocs de referència utilitzats durant la realització del projecte 'MyWall'.

Referències

[1] Using the Android Toolbar (ActionBar). http://www.vogella.com/tutorials/android.html

[2] Android Tab Layout with Swipeable Views. http://www.androidhive.info/

[3] Adding and Handling Actions. http://developer.android.com/training

[4] ActionButtons Android. http://developer.android.com/training

[5] Scanning Via Intent - QR. https://github.com/zxing/zxing/wiki/Scanning-Via-Intent

[6] Toolbar structure (best practices). http://www.google.com/design/spec/layout/structure.html#structure-toolbars

[7] Android, send "POST" JSON Data to Server. http://hmkcode.com/android-send-json-data-to-server/

[8] HTTP Status Code Definitions. https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

[9] Parse into Java Models for Android. http://www.androidhive.info/

[10] Parse into Java Models for Android. http://kylewbanks.com/

[11] Parse into Java Models for Android. http://javapapers.com/

[12] Use ListView with Android. http://www.androidhive.info/

[13] Regular expression operations - Python. https://docs.python.org/2/library/re.html

[14] Color design for Android. http://desarrollador-android.com/material-design/diseno-material-design/estilo/color/

[15] Django REST Framework. http://www.django-rest-framework.org/

[16] Django Project. https://www.djangoproject.com/

[17] PreferenceActivity for Android. http://developer.android.com/reference/android/preference/PreferenceActivity.html

[18] Climbing history and evolution. http://os2o.com/blog/la-escalada-en-roca-historia-evolucion-y-modalidades/

[19] JSON introduction http://www.json.org/json-es.html

[20] Django REST Framework Token Authentication http://www.django-rest-framework.org/api-guide/authentication/#tokenauthentication

[21] Photo collage maker http://www.photocollage.com/collage/

8 Annexos

8.1 Implementació pas a pas

8.1.1 Instal·lar, actualitzar i 'securitzar' Centos 7

S'ha contractat un servidor virtual a l'empresa OVH i clonat el disc dur amb una imatge preinstal·lada de Centos 7.
Un cop el servidor es troba operatiu i amb connexió a Internet, accedim mitjançant SSH per actualitzar els paquets de sistema a l'ultima versió disponible.

```
yum clean all
yum list all
yum update
```

Un cop llistats els paquets a actualitzar acceptem l'actualització polsant la tecla 'y'. Veure figura 30

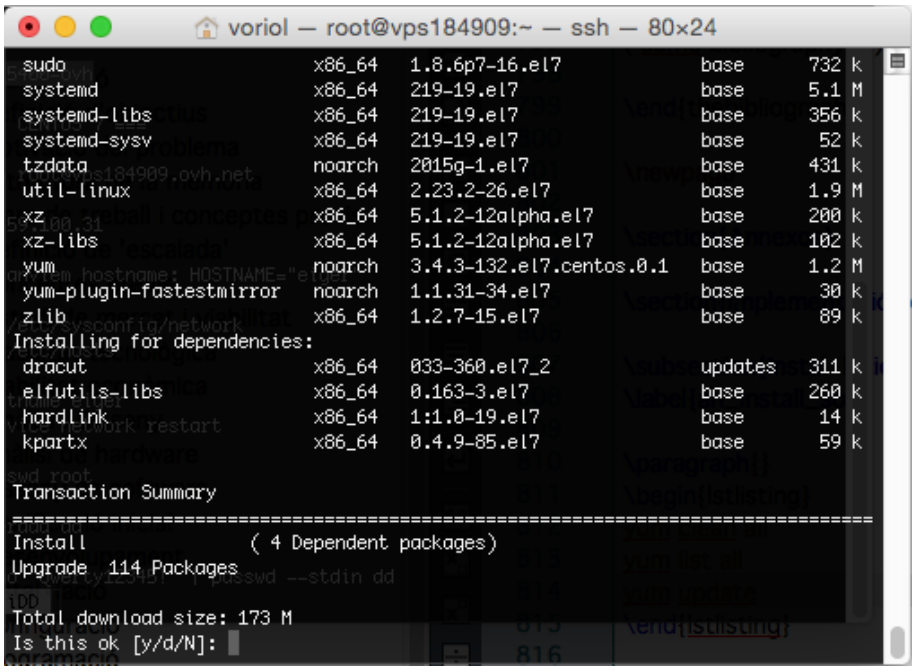

Figura 30: Resultat de 'yum update', per a 'upgradejar' els paquets hem de polsar la tecla 'y'

Per temes de seguretat s'ha modificat la contrasenya de root i creat un usuari per a l'administració del programari sense poders de 'super usuari'

```
passwd root
useradd dd
passwd dd
```

S'ha deshabilitat l'accés a root directament i mitjançant SSH donada la quantitat d'atacs que es reben per aquesta via.

```
echo "" >> /etc/ssh/sshd_config
echo "PermitRootLogin no" >> /etc/ssh/sshd_config
```

Per a què els canvis a la configuració del servei SSH tinguin validesa s'ha de reiniciar el servei, per a no perdre la sessió en cas d'haver comès un error a l'escriure sobre el fitxer de configuració efectuarem un 'kill HUP' del PID del servei SSH.

```
ps -ef |grep sshd
kill -HUP <pid>
```

8.1.2 Instal·lar MariaDB

El primer que hem de fer és afegir el 'repositori' de MariaDB al sistema, per això crearem el fitxer /etc/yum.repos.d/MariaDB.repo

```
vi /etc/yum.repos.d/MariaDB.repo
```

Amb el següent contingut:

```
[mariadb]
name = MariaDB
baseurl = http://yum.mariadb.org/10.0/centos7-amd64
gpgkey=https://yum.mariadb.org/RPM-GPG-KEY-MariaDB
gpgcheck=1
```

Un cop afegit el nou 'repositori' podem instal·lar l'ultima versió existent de MariaDB.

```
yum clean all
yum install MariaDB-server MariaDB-client
```

Després d'instal·lar MariaDB aixequem el servei mysql i l'afegim a l'arranc del sistema.

```
service mysql start
chkconfig mysql on
```

Modifiquem la contrasenya de l'usuari root (a nivell MySQL)

```
mysqladmin -u root password 'nou_passwd'
```

Amb això, el servidor de base de dades queda instal·lat, configurat i escoltant peticions.

8.1.3 Instal·lar Django Framework i dependències

La instal·lació de Django Framework s'ha fet a partir de pip (Python Package Index), un sistema de paquets utilitzat per instal·lar i administrar paquets de software escrits en Python. Primer de tot instal·larem el pip:

```
yum install python-pip
```

Un cop instal·lat el gestor de paquets de Python utilitzem la seva interfície per a instal·lar Django Framework:

```
pip install Django
```

Posteriorment validem la instal·lació de Django Framework amb la comanda:

```
[root@vps184909 dd]# django-admin --version
1.8.3
```

Un cop instal·lat el framework de Django instal·larem dependències de MySQL:

```
yum install MySQL-python
```

8.1.4 Instal·lar Django REST Framework

La instal·lació de Django REST framework i les seves dependències s'ha dut a terme mitjançant pip:

```
pip install djangorestframework
pip install markdown
pip install django-filter
```

8.1.5 Convertir 'runserver' a servei de sistema mitjançant systemctl

Per a convertir el servidor de desenvolupament de Django en un servei mitjançant systemctl només és necessari crear el següent fitxer de configuració amb el contingut:

```
[root@vps184909 ~]# cat /etc/systemd/system/runserver.service
[Unit]
Description=Django debug under 37.59.100.31:8080
After=network.target

[Service]
ExecStart=/bin/python /home/dd/DeuDits/djangoDD/manage.py runserver 37.59
KillMode=process
Restart=on-failure

[Install]
WantedBy=default.target
```

8.1.6 Crear un nou projecte i aplicació a Django

El primer pas és crear un nou projecte Django:

```
django-admin startproject djangoDD
```

A continuació s'ha de crear la base de dades que allotjarà la nova aplicació de Django:

```
mysql -u root -p -e "create database DeuDitsDB"
```

Un cop creada la base de dades, s'ha de relacionar el projecte de Django amb la base de dades de Maria DB, per això inclourem la definició DATABASES al fitxer settings.py (djangoDD/djangoDD/settings.py) del projecte:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'DeuDitsDB',
        'USER': 'root',
        'PASSWORD': 'contrassenya_root_mariaDB'
    }
}
```

Per a que Django pugui utilitzar REST framework és necessari afegir-lo al vector 'INSTALLED_APPS' del fitxer settings.py (djangoDD/djangoDD/settings.py) del projecte, s'afegirà la línia següent:

```
'rest_framework'
```

Ara és el moment de crear la nova aplicació:

```
python manage.py startapp ranking
```

Un cop creada l'aplicació 'ranking' s'ha d'afegir al vector 'INSTALLED_APPS' del fitxer settings.py (djangoDD/djangoDD/settings.py) del projecte; s'afegirà la línia següent:

```
'ranking'
```

8.1.7 Crear un nou model de dades a Django

Un cop creat el projecte i la nova aplicació és necessari definir l'estructura o model de dades, per això s'ha d'editar el fitxer models.py (djangoDD/ranking/models.py) inclòs en el directori de l'aplicació Django creada anteriorment:

```
from django.db import models

# Create your models here.

class Bloc(models.Model):
    ROSA = 'R'
    GROC = 'G'
    TARONJA ='T'
    VERMELL = 'V'
    MARRO = 'M'
    VERD = 'VE'
    BLAU = 'B'
    LILA = 'L'
    NEGRE = 'N'

    TAULA_COLORS = (
        (ROSA, 'rosa'),
        (GROC, 'groc'),
        (TARONJA, 'taronja'),
        (VERMELL, 'vermell'),
        (MARRO, 'marro'),
        (VERD, 'verd'),
        (BLAU, 'blau'),
        (LILA, 'lila'),
```

```
        (NEGRE, 'negre'),
    )

    zona = models.ForeignKey('Zona')
    nom = models.CharField(max_length=20, blank=False, null=False)
    grau = models.CharField(max_length=2, choices=TAULA_COLORS, default=B
    data_creacio = models.DateTimeField(auto_now_add=True)
    actiu = models.BooleanField(default=True)

    punts =  models.IntegerField()

    def save(self, *args, **kwargs):
        puntuacio = {'R':40, 'G':50, 'T':60, 'V':80, 'M':100, 'VE':120,
        punts = puntuacio[self.grau]

        self.punts = punts
        super(Bloc, self).save(*args, **kwargs)


class Encadena(models.Model):
    soci = models.ForeignKey('Soci')
    bloc = models.ForeignKey('Bloc')
    n_intents = models.IntegerField()
    data_encadene = models.DateTimeField(auto_now_add=True)
    comentaris = models.CharField(max_length=150, blank=True, null=True)
    punts =  models.IntegerField()

    class Meta:
        unique_together = ['soci', 'bloc']

    def save(self, *args, **kwargs):
        intents = self.n_intents
        if intents > 3: intents = 4
        punts = self.bloc.punts -(1 -intents) *10

        self.punts = punts
        super(Encadena, self).save(*args, **kwargs)


class Soci(models.Model):
    user = models.ForeignKey('auth.User', related_name='s_user')
    nom = models.CharField(max_length=20, blank=False, null=False)
    cognom = models.CharField(max_length=20, blank=False, null=False)
    cognom2 = models.CharField(max_length=20, blank=False, null=False)
    email = models.CharField(max_length=20, blank=False, null=False)
    telefon = models.CharField(max_length=20, blank=False, null=False)
```

```
    data_naixement = models.DateField()
    data_inscripcio = models.DateTimeField(auto_now_add=True)

    img_perfil = models.ImageField(upload_to='profile', default=None)

    def __unicode__(self):
            return '%s %s' % (self.nom, self.cognom)


class Zona(models.Model):
    nom = models.CharField(max_length=20, blank=False, null=False)
    ubicacio = models.CharField(max_length=20, blank=False, null=False)


class Favorits(models.Model):
    soci = models.ForeignKey('Soci', related_name='f_soci')
    favorits = models.ManyToManyField(Soci, related_name='f_favorits')
```

Un cop definit el model en el fitxer models.py és necessari sincronitzar Django amb la base de dades per a escriure els canvis en la base de dades MySQL creada anteriorment.

```
python manage.py makemigrations ranking
python manage.py sqlmigrate ranking 0001
python manage.py migrate
```

8.1.8 Definir URLs a Django

Ara que tenim definit el model de dades és necessari afegir les diferents URLs de l'aplicació a la configuració de Django, per això és necessari editar el fitxer urls.py (djangoDD/ranking/urls.py) ubicat en el directori de l'aplicació. En el cas de l'aplicació 'MyWall' les URLs utilitzades són les següents:

```
from django.conf.urls import url

from . import views

urlpatterns =
    # soci urls
    url(r'^soci/\$', views.soci, name='soci'),
    url(r'^soci_pk/\$', views.soci_pk, name='soci_pk'),
    url(r'^soci_sn/(?P<id_bloc>[0-9]{5})/\$', views.soci_sn, name='soci_s

    # bloc urls
    url(r'^bloc/\$', views.bloc, name='bloc'),
    url(r'^bloc_sn/(?P<id_soci>[0-9]{5})/\$', views.bloc_sn, name='bloc_s
```

```
    url(r'^comentaris/(?P<id_bloc>[0-9]{5})/\$', views.comentaris, name=
    url(r'^encadena/\$', views.encadena, name='encadena'),

    # favorits urls
    url(r'^favorits/(?P<id_soci>[0-9]{5})/\$', views.favorits, name='favo
    url(r'^add_favorits/\$', views.add_favorits, name='add_favorits'),
    url(r'^del_favorits/\$', views.del_favorits, name='del_favorits'),

    # upload profile urls
    url(r'^upload_image/(?P<id_soci>[0-9]{5})/\$', views.upload_image, na
    url(r'^upload_passwd/\$', views.upload_passwd, name='upload_image')

]
```

8.1.9 Definir vistes a Django

Un cop 'mapejades' les vistes en el fitxer de URLs s'ha de definir cadascuna en el fitxer views.py inclòs en el directori de la nostra aplicació Django. A continuació, es mostra el contingut del fitxer views.py (djangoDD/ranking/views.py) inclòs en el directori de l'aplicació Django. Inclou els diferents imports per a poder capturar els mètodes HTTP utilitzats a la request, accedir als models de dades i 'serialitzar-les'.

```
from django.views.generic.detail import DetailView
from django.views.generic.list import ListView
from django.shortcuts import render
from django.http import JsonResponse
from django.http import HttpResponse

from django.template import RequestContext, loader
from django.db.models import Sum

from django.http import HttpResponseRedirect
from django.core.urlresolvers import reverse
from django.shortcuts import render_to_response

from rest_framework import status
from rest_framework.response import Response
from rest_framework.decorators import api_view, authentication_classes, p
from rest_framework.authentication import SessionAuthentication, BasicAu
from rest_framework.permissions import IsAuthenticated

from forms import UploadFileForm

from ranking.serializers import *
```

```
import json

# Create your views here.

from .models import Soci
from .models import Bloc
from .models import Encadena

######################
# auth zone: vista que crea un token automaticament al donar d'alta un no
######################

from django.conf import settings
from django.db.models.signals import post_save
from django.dispatch import receiver

from rest_framework.authtoken.models import Token

@receiver(post_save, sender=settings.AUTH_USER_MODEL)
def create_auth_token(sender, instance=None, created=False, **kwargs):
    if created:
        Token.objects.create(user=instance)

######################
# end auth zone
######################

# funcio que retorna el llistat de socis
@api_view(['GET'])
@authentication_classes((SessionAuthentication, BasicAuthentication, Tok
@permission_classes((IsAuthenticated,))
def soci(request):
    soci_list = Soci.objects.all()
    soci_serialized = SociSerializer(soci_list, many=True)

    return Response(soci_serialized.data)


# funcio que retorna els socis que han encadenat un bloc en format "Soci"
@api_view(['GET'])
@authentication_classes((SessionAuthentication, BasicAuthentication, Tok
@permission_classes((IsAuthenticated,))
def soci_sn(request, id_bloc):
    soci_sn_list = Encadena.objects.filter(bloc__pk=id_bloc).order_by('-c
    soci_sn_serialized = SociByBlocSerializer(soci_sn_list, many=True)
```

```
    return Response(soci_sn_serialized.data)


# funcio que retorna la pk d'un soci a partir d'el seu token
@api_view(['GET'])
@authentication_classes((SessionAuthentication, BasicAuthentication, Tok
@permission_classes((IsAuthenticated,))
def soci_pk(request):
    soci_pk_list = Soci.objects.filter(user=request.user)
    soci_pk_serialized = SociPkSerializer(soci_pk_list, many=True)

    return Response(soci_pk_serialized.data)


# funcio que retorna el llistat de blocs
@api_view(['GET'])
@authentication_classes((SessionAuthentication, BasicAuthentication, Tok
@permission_classes((IsAuthenticated,))
def bloc(request):
    bloc_list = Bloc.objects.all().order_by('-actiu')
    bloc_serialized = BlocSerializer(bloc_list, many=True)

    return Response(bloc_serialized.data)


# funcio que retorna els blocs que encadena un soci en format "Bloc"
@api_view(['GET'])
@authentication_classes((SessionAuthentication, BasicAuthentication, Tok
@permission_classes((IsAuthenticated,))
def bloc_sn(request, id_soci):
    bloc_sn_list = Encadena.objects.filter(soci=id_soci).order_by('-data_
    bloc_sn_serialized = BlocBySociSerializer(bloc_sn_list, many=True)

    return Response(bloc_sn_serialized.data)


# funcio que retorna els comentris d'un bloc
@api_view(['GET'])
@authentication_classes((SessionAuthentication, BasicAuthentication, Tok
@permission_classes((IsAuthenticated,))
def comentaris(request, id_bloc):
    comentaris_list = Encadena.objects.filter(bloc=id_bloc).order_by('-da
    comentaris_serialized = ComentarisSerializer(comentaris_list, many=T

    return Response(comentaris_serialized.data)
```

```
# funcio que registra un nou "encadene"
@api_view(['POST'])
@authentication_classes((SessionAuthentication, BasicAuthentication, Tok
@permission_classes((IsAuthenticated,))
def encadena(request):
    serializer = EncadenaSerializer(favorits, data=request.data)

    if serializer.is_valid():
        serializer.save()
        return Response(serializer.data, status=status.HTTP_201_CREATED)

    return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST


# funcio que retorna els socis favorits del soci id_soci
@api_view(['GET'])
@authentication_classes((SessionAuthentication, BasicAuthentication, Tok
@permission_classes((IsAuthenticated,))
def favorits(request, id_soci):
    favorit_obj = Favorits.objects.get(soci=id_soci)

    soci_list = favorit_obj.favorits.all()
    soci_serialized = SociSerializer(soci_list, many=True)

    return Response(soci_serialized.data)


# funcio que permet afegir un favorit
@api_view(['POST'])
@authentication_classes((SessionAuthentication, BasicAuthentication, Tok
@permission_classes((IsAuthenticated,))
def add_favorits(request):
    serializer = FavoritsSerializer(data=request.data)

    if serializer.is_valid():
        soci = int(serializer.data['soci'])
        favorit = int(serializer.data['favorits'][0])

        Favorits.objects.get(soci=soci).favorits.add(favorit)

        return Response(serializer.data, status=status.HTTP_200_OK)

    return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST
```

```
# funcio que permet eliminar un favorit
@api_view(['POST'])
@authentication_classes((SessionAuthentication, BasicAuthentication, Tok
@permission_classes((IsAuthenticated,))
def del_favorits(request):
    serializer = FavoritsSerializer(data=request.data)

    if serializer.is_valid():
        soci = int(serializer.data['soci'])
        favorit = int(serializer.data['favorits'][0])

        Favorits.objects.get(soci=soci).favorits.remove(favorit)

        return Response(serializer.data, status=status.HTTP_200_OK)

    return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST


# funcio que permet pujar una nova imatge
@api_view(['POST'])
@authentication_classes((SessionAuthentication, BasicAuthentication, Tok
@permission_classes((IsAuthenticated,))
def upload_image(request, id_soci):
    if request.method == 'POST':
        form = UploadFileForm(request.POST, request.FILES)

        if form.is_valid():
            new_file = Soci(file = request.FILES['file'])
            new_file.save()

            return HttpResponseRedirect(reverse('main:home'))
    else:
        form = UploadFileForm()

    data = {'form': form}
    return render_to_response('main/index.html', data, context_instance=


# funcio que permet canviar el passwd d'un usuari
@api_view(['POST'])
@authentication_classes((SessionAuthentication, BasicAuthentication, Tok
@permission_classes((IsAuthenticated,))
def upload_passwd(request):
    serializer = UploadPasswdSerializer(data=request.data)

    if serializer.is_valid():
```

```
            serializer.save()
            return Response(serializer.data, status=status.HTTP_201_CREATED)

        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST
```

8.1.10 'Serialitzar' objectes a Django

Un cop instal·lat el framework només és necessari crear el fitxer serializers.py (djangoDD/ranking/serializers.py) on definirem quines dades del model volem incloure en el JSON per a posteriorment retornar-lo en la HTTP response.

```
from rest_framework import serializers
from django.db.models import Sum

from ranking.models import *


# Serializers define the API representation.

class SociSerializer(serializers.ModelSerializer):
    puntuacio = serializers.SerializerMethodField('getPunts')

    def getPunts(self, instance):
        puntuacio = Encadena.objects.filter(soci=instance).aggregate(Sum(
        if not puntuacio: puntuacio = 0
        return puntuacio

    class Meta:
        model = Soci
        fields = ('pk', 'nom', 'cognom', 'img_perfil', 'puntuacio')


class SociByBlocSerializer(serializers.ModelSerializer):
    pk = serializers.SerializerMethodField('getPkSoci')
    nom = serializers.SerializerMethodField('getNomSoci')
    cognom = serializers.SerializerMethodField('getCognomSoci')
    puntuacio = serializers.SerializerMethodField('getPuntsSoci')

    def getPkSoci(self, instance):
        return instance.soci.pk

    def getNomSoci(self, instance):
        return instance.soci.nom

    def getCognomSoci(self, instance):
        return instance.soci.cognom
```

```
    def getPuntsSoci(self, instance):
        puntuacio = Encadena.objects.filter(soci=instance.soci).aggregate
        if not puntuacio: puntuacio = 0
        return puntuacio

    class Meta:
        model = Soci
        fields = ('pk', 'nom', 'cognom', 'puntuacio')


class SociPkSerializer(serializers.ModelSerializer):
    class Meta:
        model = Soci
        fields = ('pk',)


class BlocSerializer(serializers.ModelSerializer):
    class Meta:
        model = Bloc
        fields = ('pk', 'nom', 'zona', 'grau', 'actiu', 'punts')


class EncadenaSerializer(serializers.ModelSerializer):
    bloc_nom = serializers.SerializerMethodField('getNomBloc')
    bloc_grau = serializers.SerializerMethodField('getGrauBloc')

    def getNomBloc(self, instance):
        return instance.bloc.nom

    def getGrauBloc(self, instance):
        return instance.bloc.grau

    class Meta:
        model = Encadena
        fields = ('bloc_nom', 'bloc_grau', 'n_intents', 'data_encadene')


class BlocBySociSerializer(serializers.ModelSerializer):
    nom = serializers.SerializerMethodField('getNomBloc')
    zona = serializers.SerializerMethodField('getZonaBloc')
    grau = serializers.SerializerMethodField('getGrauBloc')
    actiu = serializers.SerializerMethodField('getActiuBloc')
    punts = serializers.SerializerMethodField('getPuntsBloc')

    def getNomBloc(self, instance):
```

```
        return instance.bloc.nom

    def getZonaBloc(self, instance):
        return instance.bloc.zona.pk

    def getGrauBloc(self, instance):
        return instance.bloc.grau

    def getActiuBloc(self, instance):
        return instance.bloc.actiu

    def getPuntsBloc(self, instance):
        return instance.bloc.punts

    class Meta:
        model = Bloc
        fields = ('pk', 'nom', 'zona', 'grau', 'actiu', 'punts')


class ComentarisSerializer(serializers.ModelSerializer):
    nom_soci = serializers.SerializerMethodField('getNomSoci')
    cognom_soci = serializers.SerializerMethodField('getCognomSoci')

    def getNomSoci(self, instance):
        return instance.soci.nom

    def getCognomSoci(self, instance):
        return instance.soci.cognom

    class Meta:
        model = Encadena
        fields = ('nom_soci', 'cognom_soci', 'comentaris', 'data_encadene


class EncadenaSerializer(serializers.ModelSerializer):
    class Meta:
        model = Encadena
        fields = ('soci', 'bloc', 'n_intents', 'comentaris')


class FavoritsSerializer(serializers.ModelSerializer):
    class Meta:
        model = Favorits
        fields = ('soci', 'favorits')
```

```
class UploadPasswdSerializer(serializers.ModelSerializer):
    class Meta:
        model = Soci
        fields = ('pk', 'passwd')
```

8.1.11 'TokenAuthentication' a Django REST

A continuació s'explicarà detalladament quins passos s'han de seguir per a que les peticions HTTP requereixin autenticació.

En primer, lloc és necessari afegir la següent línia en el vector INSTALLED_APPS del settings.py (djangoDD/djangoDD/settings.py) del projecte:

```
'rest_framework.authtoken'
```

Després, en el mateix fitxer s'ha de crear el vector REST_FRAMEWORK on s'afegirà el mètode d'autenticació utilitzat.

```
REST_FRAMEWORK = {
        'DEFAULT_AUTHENTICATION_CLASSES': (
                'rest_framework.authentication.BasicAuthentication',
                'rest_framework.authentication.TokenAuthentication',
                'rest_framework.authentication.SessionAuthentication',
        )
}
```

Un cop modificat el fitxer settings.py és necessari sincronitzar la base de dades per a que els canvis quedin aplicats. Executem:

```
python manage.py syncdb
```

Un cop modificat el settings.py és necessari declarar l'autenticació de REST Framework en el fitxer urls.py (djangoDD/django/urls.py) del projecte de Django afegint les següents línies a la taula de 'urlpatterns':

```
url(r'^api-auth/', include('rest_framework.urls', namespace='rest_framewo
url(r'^api-token-auth/', views.obtain_auth_token)
```

Per a què els Tokens es creïn de forma dinàmica afegirem el següent codi en el fitxer views.py (djangoDD/ranking/views.py) de l'aplicació:

```
from django.conf import settings
from django.db.models.signals import post_save
from django.dispatch import receiver
```

```
from rest_framework.authtoken.models import Token

@receiver(post_save, sender=settings.AUTH_USER_MODEL)
def create_auth_token(sender, instance=None, created=False, **kwargs):
    if created:
        Token.objects.create(user=instance)
```

Quan usem TokenAuthentication, necessitem un mecanisme per a que els clients puguin obtenir el seu token a partir d'un usuari i una contrasenya vàlids. REST framework inclou aquesta funcionalitat, només és necessari afegir el següent codi al fitxer urls.py (djangoDD/djangoDD/urls.py) del projecte Django:

```
from rest_framework.authtoken import views
urlpatterns += [
    url(r'^api-token-auth/', views.obtain_auth_token)
]
```

Això és suficient per utilitzar 'TokenAuthentication' inclòs en Django REST Framework.

8.1.12 Configuració de fitxers estàtics a Django

A continuació s'explicarà com configurar Django per poder utilitzar fitxers estàtics. En el cas de l'aplicació 'MyWall' s'han utilitzat els continguts estàtics per emmagatzemar les imatges de perfil dels usuaris.

S'han d'afegir les següents línies en el fitxer settings.py (djangoDD/djangoDD/-settings.py) del projecte Django:

```
STATIC_URL = '/static/'

STATIC_ROOT = BASE_DIR +'/static'
MEDIA_ROOT = BASE_DIR +'/media/'
MEDIA_URL = '/media/'
```

Un cop definits els directoris en el settings.py és necessari donar accés als recursos a través del fitxer urls.py del projecte de Django sumant les següents opcions a la taula de 'urlpatterns':

```
+ static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT) + static(
```

Amb això queden definides les rutes per a fitxers estàtics del projecte de Django.

8.1.13 Configuració de la pàgina d'administració a Django

Per configurar la pàgina d'administració de Django només és necessari afegir els models que es volen administrar i indicar els camps que es volen mostrar per a cadascun en el fitxer admin.py (djangoDD/ranking/admin.py) de l'aplicació de Django.

```
MacBook-de-victor-oriol:memoria voriol$ cat ../djangoDD/ranking/
from django.contrib import admin

# Register your models here.

from .models import *

class SociAdmin(admin.ModelAdmin):
list_display = ('cognom', 'nom', 'email', 'telefon')

class BlocAdmin(admin.ModelAdmin):
def get_nom_zona(self, obj):
return obj.zona.nom

list_display = ('nom', 'get_nom_zona', 'grau')

class ZonaAdmin(admin.ModelAdmin):
list_display = ('nom', 'ubicacio')

admin.site.register(Soci, SociAdmin)
admin.site.register(Bloc, BlocAdmin)
admin.site.register(Zona, ZonaAdmin)
```

Els rols d'usuari i permisos vénen donats per la gestió d'usuaris del projecte Django. Es poden modificar des de la interfície web d'administració amb l'usuari admin (super user) de Django.