Treball de Fi de Grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

APLICACIÓ MULTIPLATAFORMA PER A LA DETECCIÓ DE LOGOS

Pol Garcia i Arenas

Tutor: Brais Cancela
Realitzat a: Departament de
Matemàtiques i Informàtica

Barcelona, 22 de juny de 2017

Índex

0. Estructura de la memòria 7

1. Introducció 8

a. Resum 8

b. Motivació 8

c. Objectius 9

2. Anàlisis 10

a. Tecnologia 10

Backend 10

Frontend 11

Extres 11

b. Requeriments 13

3. Disseny Funcional 16

a. App 16

b. Base de dades 16

c. Estructura de l'aplicació 19

Com s'estructura el codi font 21

d. Estructura del servidor 22

4. Disseny gràfic 24

5. Implementació 33

a. Administració Servidor Django 33

Crear esdeveniment 36

b. Aplicació Ionic 42

Controladors 42

Compilació 54

6. Continuïtat del projecte 56

7. Conclusions 59

Abstract

Nowadays, everyone has a mobile device. The current society is too obsessed with the smallscreens and they stop paying attention to what is actually happening around them.

The goal of this project is to create a multi-platform application that allows the user to achieve some type of reward when making a photograph with their mobile device. People will only be able to access these rewards by attending certain events in the city or going to certain stops distributed around the map. These stops will also return a small reward that will be available to request for a few hours.At all times the user must use their own account and, if desired, they may change their data. The interface must be simple and intuitive.

This memory will explain concepts about the structure of the project, its interface and its controllers.
Finally, we will talk about what would be needed in order to improve this app in the market.

Resum

A dia d'avui, totes les persones disposen d'un dispositiu mòbil. La societat actual esta massa obsessionada amb les petites pantalles i deixen de parar atenció als esdeveniments que tenim davant.

L'objectiu d'aquest projecte és crear una aplicació multi-plataforma que permeti a l'usuari, a través del seu dispositiu mòbil, aconseguir algun tipus de recompensa al realitzar una fotografia amb el seu dispositiu. Nomes podran accedir a elles assistint a determinats esdeveniments de la ciutat o anar a les parades distribuïdes pel mapa. Aquestes parades també retornaran una petita recompensa que es podrà sol•licitar de nou al cap d'unes hores.
En tot moment l'usuari haurà d'utilitzar un compte propi i, en el cas que es desitgi, podrà canviar les seves dades. La interfície ha de ser simple i intuïtiva.

En aquesta memòria s'explicaran conceptes sobre l'estructura del projecte, la seva interfície i els seus controladors.
Finalment es parlarà sobre les coses a millorar cara al mercat.

0. Estructura de la memòria

1. Introducció: Breu resum del projecte escollit, les motivacions de les que neix i els objectius que s'han fixat.
2. Anàlisis: Apartat dedicat a l'estudi més detallat dels requeriments concrets de l'aplicació, la tecnologia triada per a assolir-ho i un primer anàlisis inicial de l'aplicació.
3. Disseny funcional: Cobreix tant el disseny de l'aplicació com el del servidor
4. Disseny gràfic: Patrons seguits per a dissenyar una interfície d'usuari intuïtiva i responsable.
5. Implementació: Detalls sobre la implementació de l'aplicació i el servidor.
6. Continuïtat del projecte: Situació de la part desenvolupada dins el marc global de l'aplicació a comercialitzar.
7. Conclusions

1. Introducció

a. Resum

Aquest projecte ha seguit l'estructura client – servidor, amb arquitectura model – vista – controlador. És una aplicació mòbil multi-plataforma amb l'objectiu de complementar i millorar l'experiència de l'usuari en qualsevol tipus d'esdeveniments amb l'ajut de components dels telèfons mòbils com la càmera i el localitzador GPS.

Els esdeveniments s'han dividit en 2 tipus:

- Limitats: Tindran una durada limitada i només es podran guanyar un cop. Realitzant una fotografia.
- Parades: No tindran una durada limitada, però sí un temps de càrrega per tornar-se a utilitzar. Només amb un click.

Els esdeveniments limitats es podran trobar en esdeveniments públics com concerts, festes majors, esdeveniments esportius... En els quals s'haurà de realitzar una fotografia al objectiu requerit per guanyar una recompensa que pot ser un val de descompte al bar de davant, per exemple.

Els esdeveniments parada es podran trobar per qualsevol lloc de la teva ciutat. Només veure'l, fent click, guanyarem una petita recompensa, equivalent al típic flyer que pots aconseguir a la porta d'un restaurant de menjar ràpid per exemple. La seva posició no canvia, però si que tenen un temps de recàrrega.

Amb l'aplicació es podrà accedir en tot moment als esdeveniments disponibles i a les recompenses aconseguides.

b. Motivació

Abans d'escollir realitzar aquesta app, ja tenia clar que, a no ser que m'agradés molt algun dels temes proposats per la universitat, volia embarcar-me en un projecte pensat per mi. Veure com una petita idea va agafant cara i ulls i es converteix en el que havies pensat que, al final, sempre canvia una mica.
Amb aquest treball es vol aprofitar la rellevància que tenen els dispositius mòbils avui en dia, els quals estan presents a totes hores en la nostre vida.

Avui dia es parla molt de que no deixem el telèfon per res i gairebé ni ens relacionem, només mirem pantalles. Qui no ha vist a un grup de nois al parc, i tots mirant les seves pantalles, callats? Doncs d'aquestes escenes va sorgir la motivació d'unir aquest concepte, amb els esdeveniments públics. Utilitzar el telèfon en cert moment al realitzar una activitat per optar a una petita recompensa.

Per aquesta idea necessitava conceptes sobre programació mòbil, dels quals tenia pocs tant d'android com d'IOs, ja que el mercat està clarament marcat i es volia arribar al màxim d'usuaris.

Sobre l'estructura client – servidor tenia bons conceptes, gràcies a l'assignatura de "Software Distribuït", on també vaig utilitzar el servidor Django, que s'utilitzarà en aquest projecte a la part de backend.

Per la part de l'arquitectura model – vista – controlador, ja tenia experiència en desenvolupament web gràcies a les pràctiques curriculars, així que no vaig dubtar en buscar eines que hem permetessin realitzar-ho d'aquesta manera.

c. Objectius

Els objectius d'aquest projecte els podem englobar en els següents:

- Una aplicació multi-plataforma que permeti:
    - Geo localitzar-te a tu i als esdeveniments propers
    - Fer / Enviar fotografies
    - Participar a esdeveniments
    - Mostrar recompenses adquirides
- Un servidor que permeti:
    - Administrar Usuaris / Esdeveniments
    - Tractar imatges
    - Guardar informació necessària

2. Anàlisis

a. Tecnologia

Un cop tenim les idees bàsiques de l'app, hem d'escollir amb què fer-ho i per què. Dividirem l'explicació en Frontend i Backend, i dins d'aquestes parlarem de les parts involucrades tant a la part del servidor com a la del client. També afegiré les eines necessàries per que funcionin totes les anteriors i poder realitzar la compilació tant per Android com per IOs.

Backend

La major part del backend està format per un servidor Django (Python). Com he dit anteriorment, aquest servidor ja l'havia utilitzat a Software Distribuït, i juntament amb que no tenia experiència en altres, i que la part de reconeixement d'imatge també seria en python, no es va dubtar en escollir-lo com a servidor definitiu.

En aquest projecte no es parlarà sobre la part del reconeixement d'imatge, però es va investigar força sobre el tema abans de decidir-se per res. Inicialment es va indagar sobre projectes com "Layar AR" . Projectes de realitat augmentada els quals et permeten realitzar un reconeixement d'imatge molt simple i, a través d'una pantalla, reproduir algun video o imatge en el mon real. Després de moltes proves es va descartar, ja que era un projecte innovador però a dia d'avui esta abandonat i és molt poc efectiu.

Llavors vam optar pel Deep Learning, ja que ens permetia ser més precisos en el reconeixement (angles, il·luminació, escalabilitat...) Després d'unes setmanes d'investigació sobre TensorFlow, Caffe, Theano... Es va decidir crear la xarxa amb Lasagne, sobre de Theano els quals funcionen en python 3.4, com el servidor.

Per la persistència de dades, treballem amb la base de dades per defecte que proporciona Django, SQLite. Ja que no hem de guardar grans quantitats d'informació és suficient.

L'última part del Backend la trobem a la part client. Està programat en angular js i forma part de "Ionic Framework", gràcies al qual l'aplicació serà multi plataforma. Parlem d'ell a l'apartat de Frontend.

Frontend

Es volia programar en multi plataforma, però no es volia perdre temps en tenir que programar en dos sistemes operatius alhora, en dos codis diferents. Així que es va investigar sobre les eines del mercat en desenvolupament híbrid i totes tenien el mateix nom, "Phonegap".

Inicialment va aparèixer Phonegap al 2008, per l'empressa "Nitobi" i l'idea va agradar molt, però necessitaven més col·laboradors i al 2011 van donar el codi a la fundació Apache, convertint el projecte en open source i canviant-li el nom a Cordova.

Gràcies a això, Cordova permet afegir plugins pròpis o de tercers per augmentar les funcionalitats de l'app. També pots crear-ne de personalitzats.

Per treballar amb cordova i, a la vegada, amb una estructura simple per treballar el frontend, es va escollit Ionic Framework.

Ionic és un framework pel desenvolupament frontend d'aplicacions mòbils que treballa la part front end html,css i el back en angular js. Ionic Framework et permet crear un projecte que funcioni en cordova, creant una estructura de carpetes organitzada i simple per treballar amb més facilitat.

Extres

Pel funcionament complert de Cordova, necessitem tenir instal·lades les següents eines:

- Nodejs 6.10.0
- Android Studio 2.3 / Xcode
- Git

No les necessitem directament, però sí per que Cordova tingui totes les dependències cobertes. L'única diferència per a la compilació per Android o per IOs, és el requisit de tenir un mac, el qual utilitzarem la comanda a Ionic per crear un fitxer xcode i amb aquest, compilar l'aplicació per a IOs, mentre que per Android, directament des de la consola d'Ionic es podrà crear l'apk.

Per indicar-ho d'una manera més simple i centralitzada, podem dir que la nostre aplicació es forma a partir de:

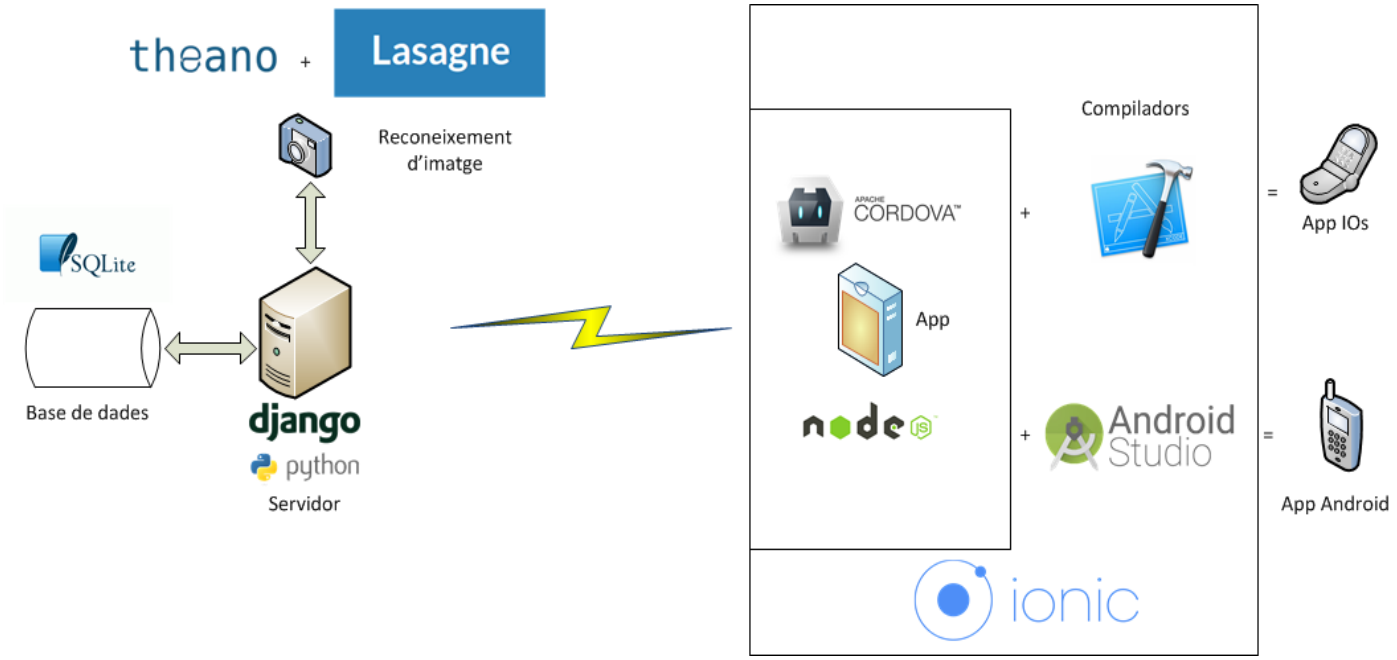


Figura 1. Estructura que dur a terme l'aplicació.

b. Requeriments

En aquest treball s'engloba la programació i maquetació de la pròpia app, la configuració i el funcionament de l'administrador i l'estructura de la base da dades.

- Ha de permetre a un Administrador:
    - Crear/Modificar/Eliminar Usuaris
    - Modificar permisos sobre l'app
    - Crear/Modificar/Eliminar Esdeveniments

- Ha de permetre a un Usuari:
    - Veure la seva posició actual al mapa
    - Veure els esdeveniments propers
    - Interactuar amb els esdeveniments
    - Modificar dades personals
    - Visualitzar recompenses

- Esdeveniments:
    - Poder guanyar una recompensa
    - Esdeveniments Limitats:
        - Data de participació limitada
        - Fotografia requerida
    - Esdeveniments Parades:
        - Esdeveniments de posició fixe
        - Temps de recàrrega

Segons els requeriments anteriors podríem generar uns cassos d'ús com els de les figures 2 i 3.

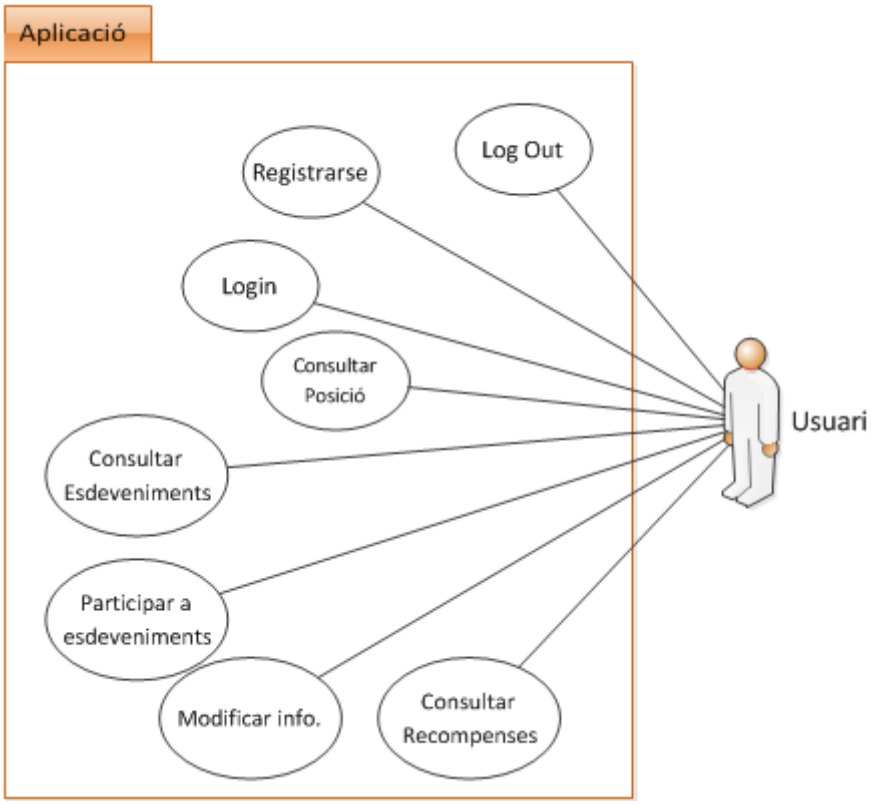


Figura 2. Petit diagrama de cassos d'ús de l'usuari

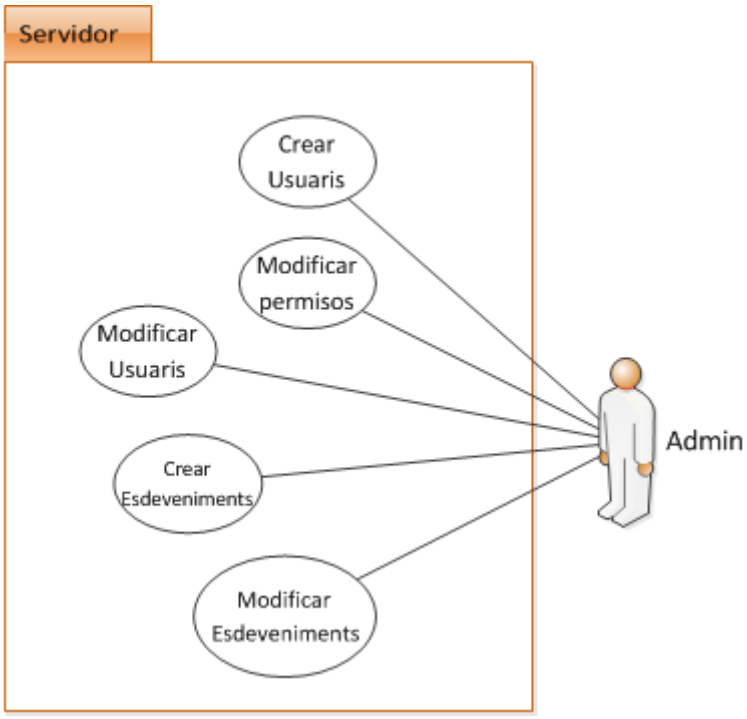


Figura 3. Petit diagrama de cassos d'ús de l'administrador

No entrarem en detalls de tots els cassos d'ús que tenim, però si que els comentarem per fer-nos una idea d'ells. Començarem amb els cassos d'ús de l'aplicació, que conté els possibles cassos que es pot veure involucrat l'usuari, mentre que continuarem amb els del Servidor, els quals fan referència a l'administrador.

Aplicació:

Com a usuari nosaltres voldrem utilitzar l'aplicació per tant necessitarem logejar-nos, en el cas de que no tinguem compte, ens hem de registrar. Sense login no es pot accedir a cap part de l'aplicació que no sigui el registre.

A partir del login, l'usuari pot accedir a tota la resta d'accions.

La primera es la de consultar la posició actual, que ens apareixerà des de la home en tot moment i ens indica la nostre posició actual i la dels esdeveniments propers.

Consultar i participar en els esdeveniments s'accedeixen des de la home de l'aplicació. Ens informa sobre els esdeveniments disponibles i ens proporciona la possibilitat de filtrar-los.

El penúltim cas d'ús és l'únic en el qual l'usuari pot modificar dades pròpies i està controlat en tot moment, amb avisos a través de popup's. L'usuari podrà modificar el seu email i la seva contrasenya.

Per últim tenim el cas d'ús de Consultar recompenses, el qual només es podrà realitzar a partir de que l'usuari hagi adquirit alguna recompensa amb anterioritat en algun esdeveniment. En cas de que no hagi guanyat cap recompensa, tindrem una llista buida.

Servidor:

Com administrador voldrem administrar la nostre aplicació i per tant els seus cassos d'ús engloba la creació/modificació d'usuaris i d'esdeveniments, incloent-hi les recompenses.

Els cassos importants serien els relacionats amb els esdeveniments, els quals ens permeten crear un esdeveniment nou i un cop el tenim, transformar-lo en limitat o en parada. Al crear l'esdeveniment també requerirem d'una recompensa per poder-lo crear. Si es comet algun error sempre es pot modificar l'esdeveniment o fins i tot canviar-lo de tipus.

Per últim també tenim un control sobre usuaris i els seus permisos. Per si es necessita un altre compte d'administrador o hi ha algun problema en alguna compte d'usuari.

3. Disseny Funcional

a. App

L'estructura de l'aplicació es divideix com en un projecte de desenvolupament web model-vista-controlador (mvc).

- Vistes: ionicApp/www/templates/.
  En aquest projecte s'utilitzen 8 vistes.
- Controladors: IonicApp/www/js/
  Per cada vista, tenim un controlador independent.
- Model: backend/python_django/restbackend/abstractApp/
  En aquest cas el trobem a la part backend de Django, ja que des d'aquí es realitzaran les consultes i ens enviarà la informació de la manera desitjada.

b. Base de dades

Podem dividir la base de dades en 2 petits blocs principals i una taula auxiliar. Els primers blocs són els que realitzen l'emmagatzematge de les dades sobre els usuaris i sobre els esdeveniments. La taula auxiliar s'utilitza per guardar les imatges que s'envien pels esdeveniments limitats. A la figura 4 podem veure les relacions entre les taules.

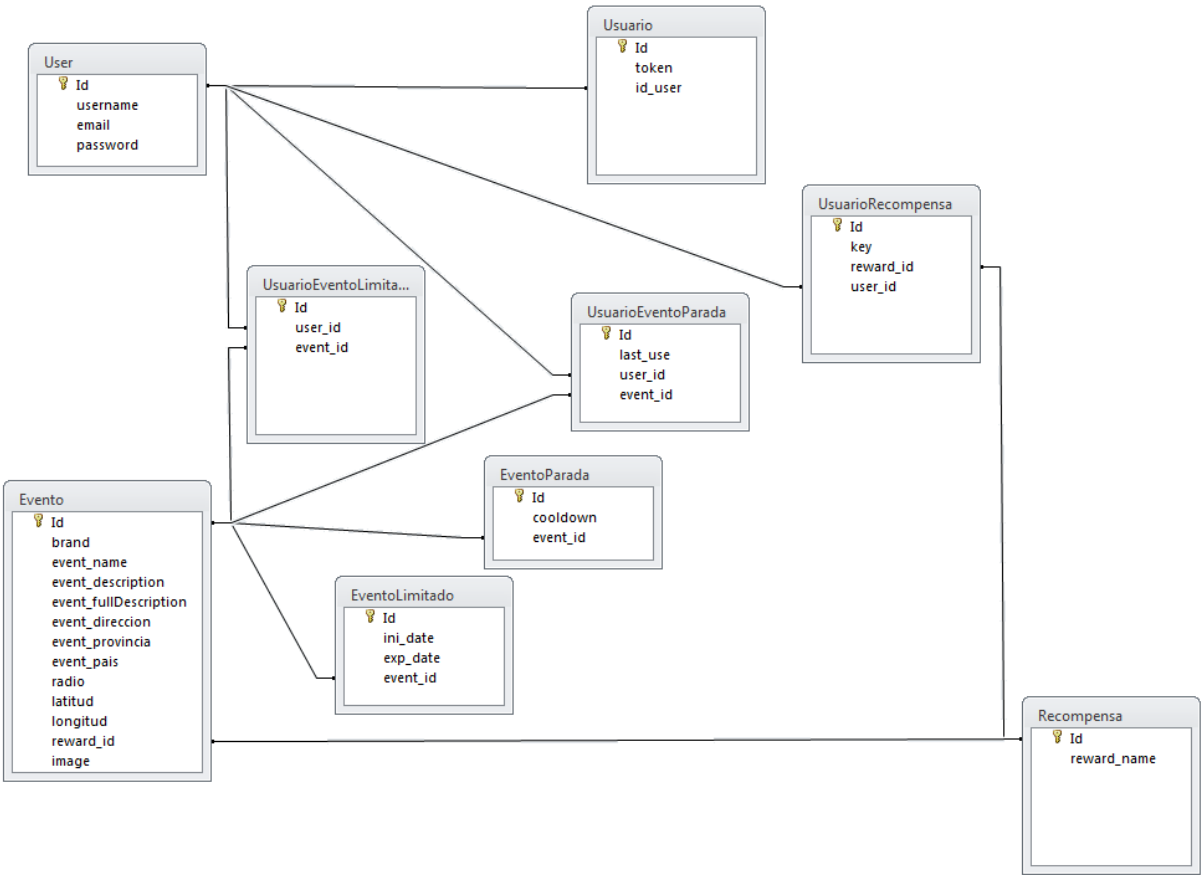


Figura 4. Diagrama de la base de dades.

Començarem comentant les taules relacionades amb els usuaris:

Usuari:

- User: És la taula per defecte per usuaris que proporciona Django. Tenim un par d'atributs que no utilitzem i per això no apareixen al diagrama, is_active i is_staff. Ens indican si un usuari està actiu i pot utilitzar-se i si som administradors. S'utilitza per guardar tota la informació bàsica de l'usuari. Django internament xifra les contrasenyes amb l'algoritme PBKDF2 amb un hash SHA256
- Usuario: Aquesta taula s'utilitza per guardar un token de l'usuari que es crea al moment de logejar, el qual es comprova a cada petició amb la copia que s'emmagatzema al telèfon.
- UsuarioRecompensa: Aquí tenim les recompenses que l'usuari vagi adquirint a mida que participi en tota mena d'esdeveniments.

Seguidament, tenim 3 taules que formen part d'Usuari, però també d'Esdeveniments. Dos d'elles són les que relacionen els diferents tipus d'esdeveniments amb cada usuari i la tercera s'encarrega de les recompenses dels esdeveniments.

- UsuarioEventoLimitado: Aquesta taula fa referència als esdeveniments limitats, els que es realitzen enviant una fotografia. Aquests esdeveniments, com el seu nom indica, duran un temps limitat i només es poden realitzar un cop. Una vegada guanyem la recompensa, no es pot repetir i d'això s'encarregarà aquesta taula.
- UsuarioEventoParada: Aqui tenim l'altre tipus d'esdeveniment. Aquest es pot repetir tantes vegades com es desitgi però té un cooldown (temps de recàrrega). Aquestes recompenses serien més simples ja que no exigeix realitzar pràcticament res apart d'un click. Aquesta taula ens servirà per poder deixar participar a l'usuari si ha passat el temps de recàrrega des de l'últim cop que ha utilitzat aquesta Parada.
- Recompensa: Gràcies a aquesta taula, tenim el control de les recompenses que existeixen i les distribuïm a partir del seu esdeveniment a l'usuari que li pertoca.

Finalment tenim les taules relacionades amb els esdeveniments:

Esdeveniments:

- Evento: En aquesta taula s'emmagatzema tota la informació bàsica que comparteixen tot tipus d'esdeveniments: La marca amb la que es treballa, el nom, unes descripcions, dades de localització i una imatge que es podria col·locar per que ho vegi el client dins de l'app. També tenim la variable radio, la qual determina el radi d'acció d'aquest esdeveniment, és a dir, als metres de distància des de els quals pots accedir i participar en l'esdeveniment.
- EventoParada: Aqui es controlen els esdeveniments Parada, amb el seu temps de recàrrega. Juntament amb UsuarioEventoParada, podrem saber si l'usuari pot utilitzar aquesta parada o encara no.
- EventoLimitado: Per altre banda, tenim els esdeveniments limitats, els quals tenen una data d'inici i un altre de finalització. Un cop la data actual es superior a la de finalització, ja no podrem participar en l'esdeveniment, ni tampoc consultar-lo.

Per acabar la repassada a la base de dades, tenim una taula auxiliar en la qual guardem totes les imatges enviades dels esdeveniments limitats, figura 5.

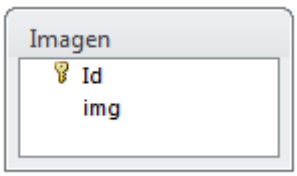


Figura 5. Taula "Imagen".

c. Estructura de l'aplicació

Aquest projecte ha sigut el primer que he fet íntegrament per mòbil multi plataforma i tampoc tenia gaire experiència en desenvolupament tant d'IOs com d'Android, per tant, aprendre l'estructura de Ionic ha sigut força fàcil ja que no tenia gaires coneixements previs i no he hagut de canviar conceptes antics. Començarem amb l'estructura de la part client, que com hem dit anteriorment, es basa en vista-controlador, figura 6.

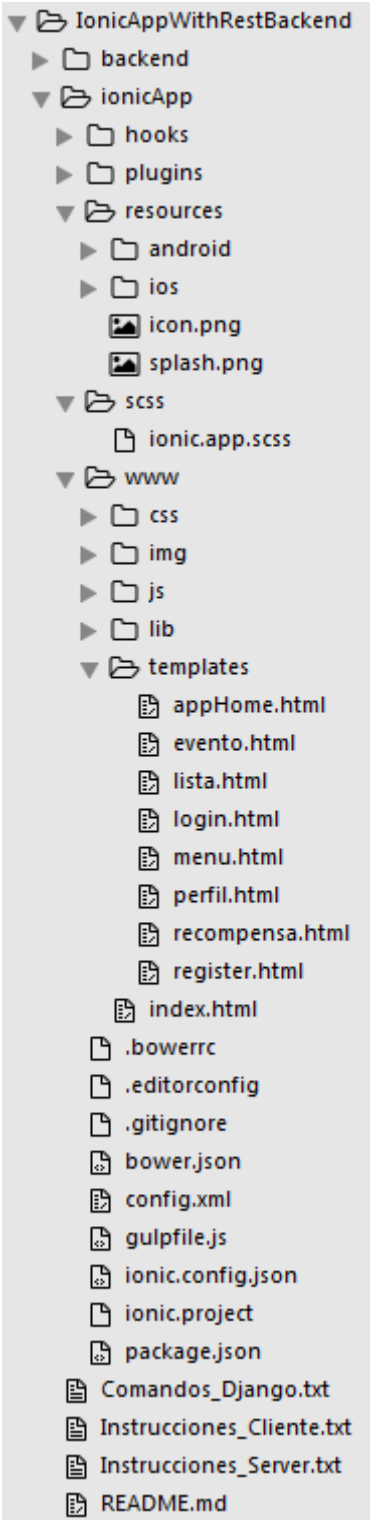


Figura 6. Estructura de carpetes de l'aplicació Ionic.

Al generar un nou projecte Ionic es genera una estructura de carpetes força intuïtiva per organitzar tot el codi, figura 3. Comentarem les seves funcions i a continuació en centrarem en la pròpia aplicació:

- Hooks/ : Aquesta carpeta s'utilitza per afegir scrips que s'executaran quan es produeixin determinats esdeveniments, com per exemple abans o després de compilar. No s'utilitzarà en aquest projecte.
- Plugins/ : Com el seu nom indica, conté els plugins que utilitzarà la nostre app, els quals ens permeten dur a terme funcionalitats extres amb els components dels mòbils. En aquest projecte utilitzem les següents funcionalitats extres:
    - Càmera: Per poder realitzar fotografies pels esdeveniments limitats.
    - Geolocalització: Detectar la posició de l'usuari i dels esdeveniments. Per això utilitzem la api de GoogleMaps, la qual es gratuïta fins a un límit de consultes diàries. Per aquest projecte de fi de grau serà suficient.
    - File: Ens permet poder llegir o escriure fitxers al telèfon.
- Resources/ : Recursos específics per a cada plataforma a la qual està dirigida l'aplicació. En aquest cas per Android i IOs.
- Scss/ : Codi scss que serà compilat a la carpeta www/css/. En aquest projecte s'ha treballat modificant directament el css.
- www/ : Conté el codi font de l'aplicació: html, css, JavaScript, imatges, etc.

Els últims fitxers contenen les dependències Bower i NodeJS, tasques de Gulp i les configuracions de Cordova i el propi projecte Ionic.

També s'han afegit unes notes informatives per poder arrancar el servidor, arrancar el client en navegador local i poder realitzar canvis d'estructura a la base de dades.

Com s'estructura el codi font

Com hem dit anteriorment, la carpeta www/ conté tot el codi font de l'aplicació i l'estructura Vista-Controlador. A continuació explicarem com s'organitza:

- Css: Aquí es guarden els fitxers css de l'app. En aquest projecte hem modificat style.css i ionic.css que el trobarem a la carpeta lib.
- Img: Conté totes les imatges utilitzades per l'aplicació.
- Lib: En aquesta carpeta tenim totes les llibreries d'Ionic i angular que utilitzarem en el projecte. Algunes venen per defecte, però hem afegit algunes extres per animacions, noves textures i efectes. Algunes d'elles son javascript i d'altres simple css.
- Templates: Aquí tenim totes les vistes de l'aplicació, en html, les quals es poden obrir en qualsevol navegador/dispositiu. Aquest projecte consta de 8 vistes:
    - appHome: Pàgina principal de l'aplicació.
    - evento: Pàgina on trobarem els detalls dels esdeveniments limitats.
    - lista: Aquí trobarem un llistat dels esdeveniments limitats disponibles. Filtrables.
    - login: Inicia de l'aplicació. És requerit logejar per poder entrar.
    - menu: És una barra desplegable, a la dreta de la pantalla, la qual ens permetrà navegar per l'aplicació.
    - perfil: Ens mostrarà el correu actual, i tindrem la possibilitat de canviar la contrasenya i el mateix correu.
    - recompensa: Aquí podrem comprovar les recompenses adquirides i els codis guanyats.
    - register: Aquesta vista és la que ens permetrà registrar-nos a l'aplicació i poder accedir.
- js: Aquesta carpeta conté tot el codi js de l'aplicació, els controladors, que seran executats pel client. Aqui realitzarem les peticions al servidor i en alguns casos algun càlcul per tractar de fer descansar al servidor.

d. Estructura del servidor

Com hem dit anteriorment, el servidor es Django, python, i ademés de realitzar la funció de Model, en l'arquitectura Model – Vista – Controlador, també ens proporciona una base de dades per defecte, SqLite, la quals ens servirà per la permanència de dades per la nostre aplicació. A la figura 7 podem veure l'estructura de carpetes de la part backend i explicarem les seves funcions.

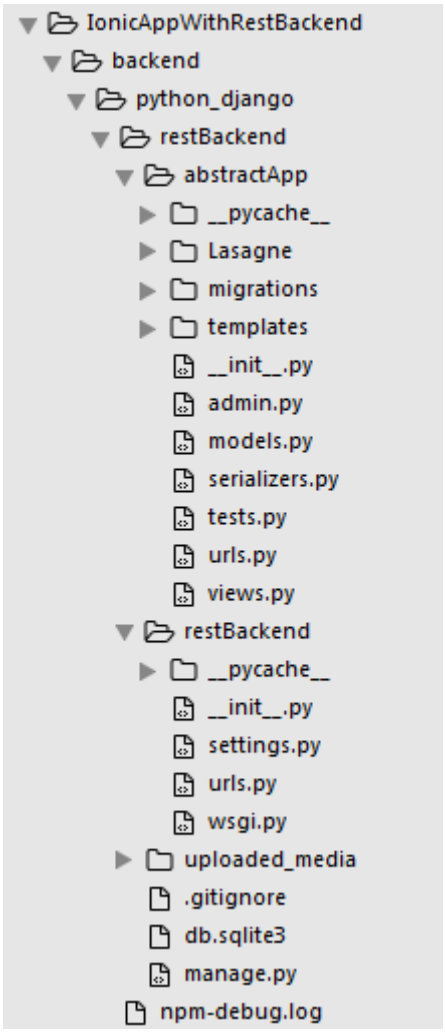


Figura 7. Estructura de carpetes de la part Django.

En aquest projecte només hem treball amb la part d'abstractApp. És la que conté les migracions de la base de dades, on s'executa la llibreria Lasagne sobre Theano pel reconeixement d'imatge, on es configura l'espai d'administrador i els models de la base de dades, entre d'altres.

- Lasagne: Lasagne i Theano són les llibreries encarregades del reconeixement d'imatge. S'ha creat un model de 10 classes amb 500 imatges per classe.

- Migrations: Aqui s'enregistren tots els canvis d'estructura de la base de dades. Si s'esborra un camp per exemple.
- Templates: Aqui es guardarien les templates d'un projecte únicament de Django. En el nostre cas, utilitzem iònic framework per tota la part d'interficie, menys la pròpia de l'administrador que es configura a admin.py
- Admin.py: Es registren els models que utilitzarem i les possibles accions que pugui fer l'administrador. Hem aplicat tota la lògica de crear/modificar Usuaris i esdeveniments.
- Models.py: Aqui tenim totes les taules que hem creat per l'aplicació. Les taules per defecte no apareixen però si que es poden fer referència. Per exemple per les relacions amb la taula user.
- Serializers.py: Son classes que ens permeten transformar dades en formats més pròpis de Django com extensions de models o querysets, però en el nostre cas no els hem utilitzat, ja que amb la pròpia sortida de views.py i el tractament d'angular.js ja ens és suficient.
- Tests.py: Fitxer per crear proves unitàries.
- Urls.py: Fitxer molt important el qual, permet al servidor poder realitzar qualsevol consulta i saber com arribar-hi. Aquesta url s'envia a través de la petició GET/POST desde els controladors d'ionic.
- Views.py: És el fitxer que realitza les consultes demandades desde ionic. És l'únic fitxer que té accés a la base de dades.

4. Disseny gràfic

Inicialment el disseny s'ha realitzat en paper. Intentant cobrir les necessitats que es tenien en el moment. Com és un projecte que ha anat evolucionant, he hagut d'aplicar diferents canvis a la interfície.
La primera variació des de l'idea inicial va ser la home, ja que en un principi tenia la idea de que la home ens informés sobre tots els esdeveniments disponibles, però aquesta opció es va canviar de posició, i s'accedeix des de la home. La nostre Home esta formada per una consulta a l'api de GoogleMaps, la qual ens indica de la nostre posició i dels esdeveniments propers. Es va pensar que d'aquesta manera cridaria més l'atenció als usuaris i és força més intuïtiu.

Login:

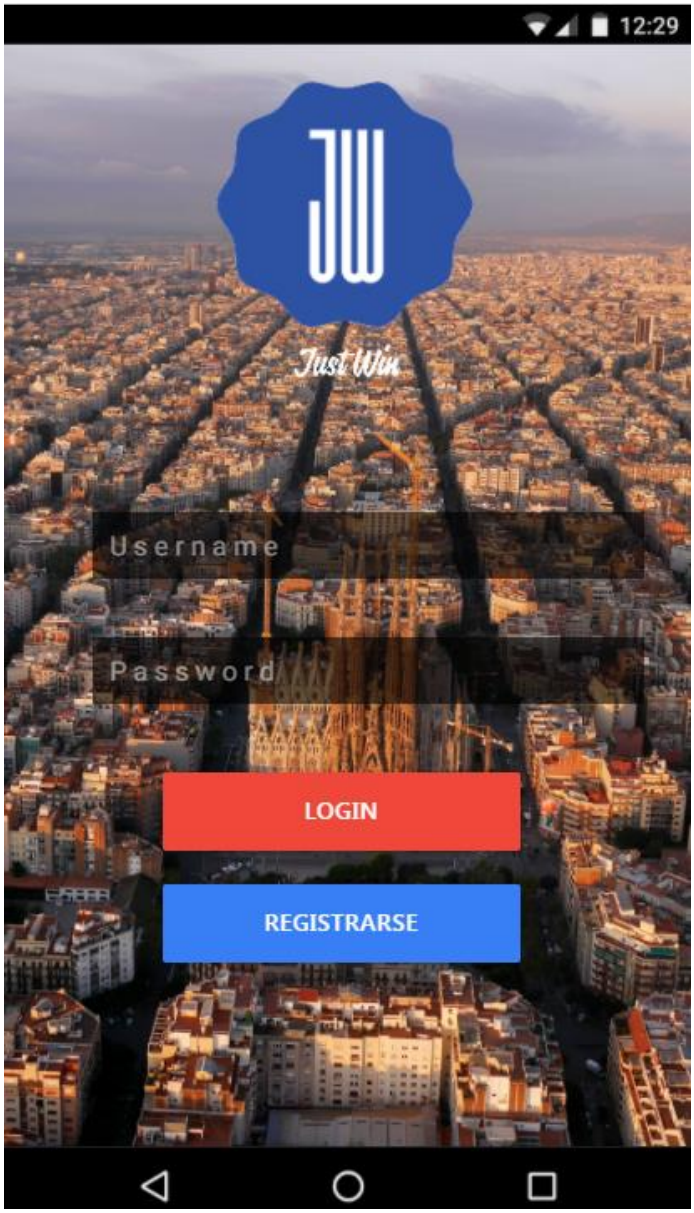


Figura 8. Pantalla de login.

A la figura 8 veiem la pàgina inicial de l'aplicació. És necessari logejar per poder accedir a l'app. Podem accedir al formulari per registrar-se o directament logejar que ens enviarà a la home.

Registre:

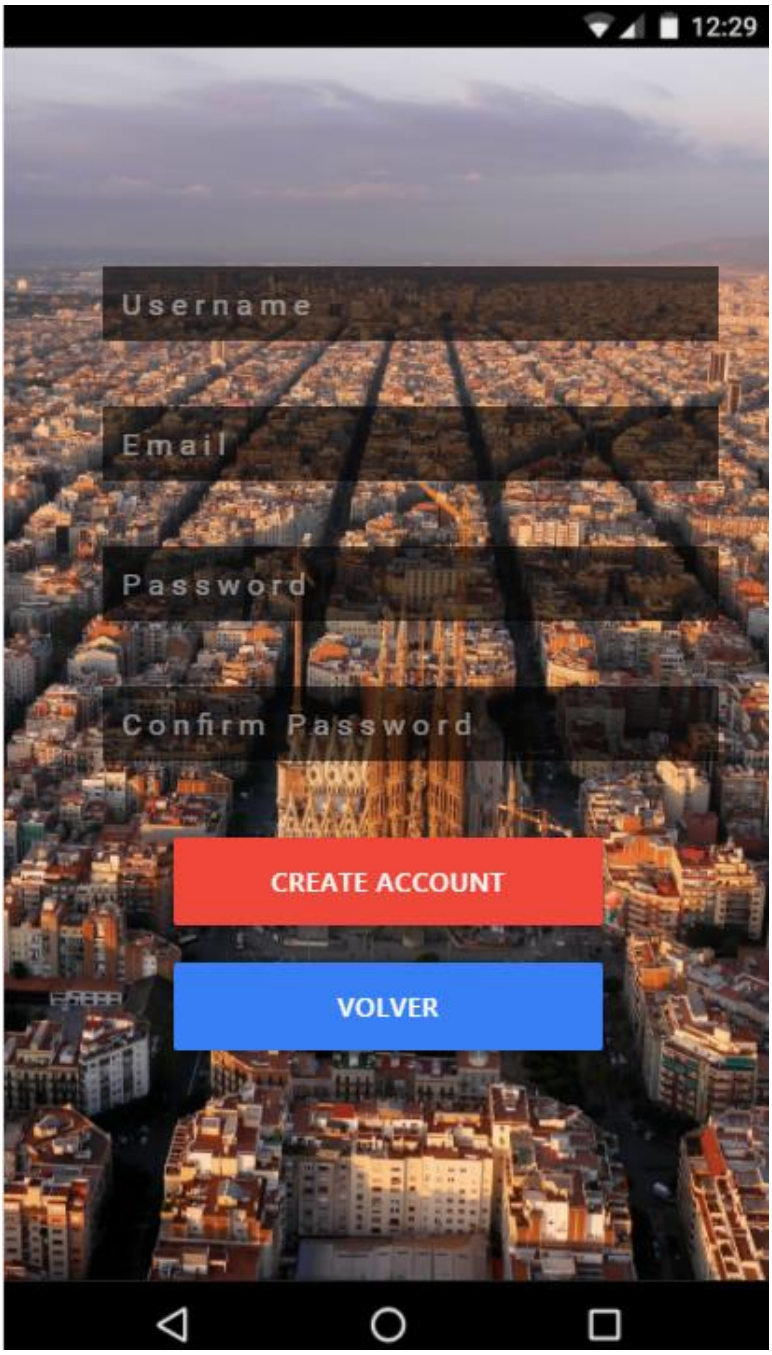


Figura 9. Pantalla de registre.

La figura 9 ens mostra el formulari per poder crear un compte dins de l'aplicació. Un cop dins, podrem realitzar alguns canvis a aquestes dades. Per aquest projecte no utilitzem el email de l'usuari per a cap funció. També podem tirar enrere, cap al login, fent click a "Volver"

Home:

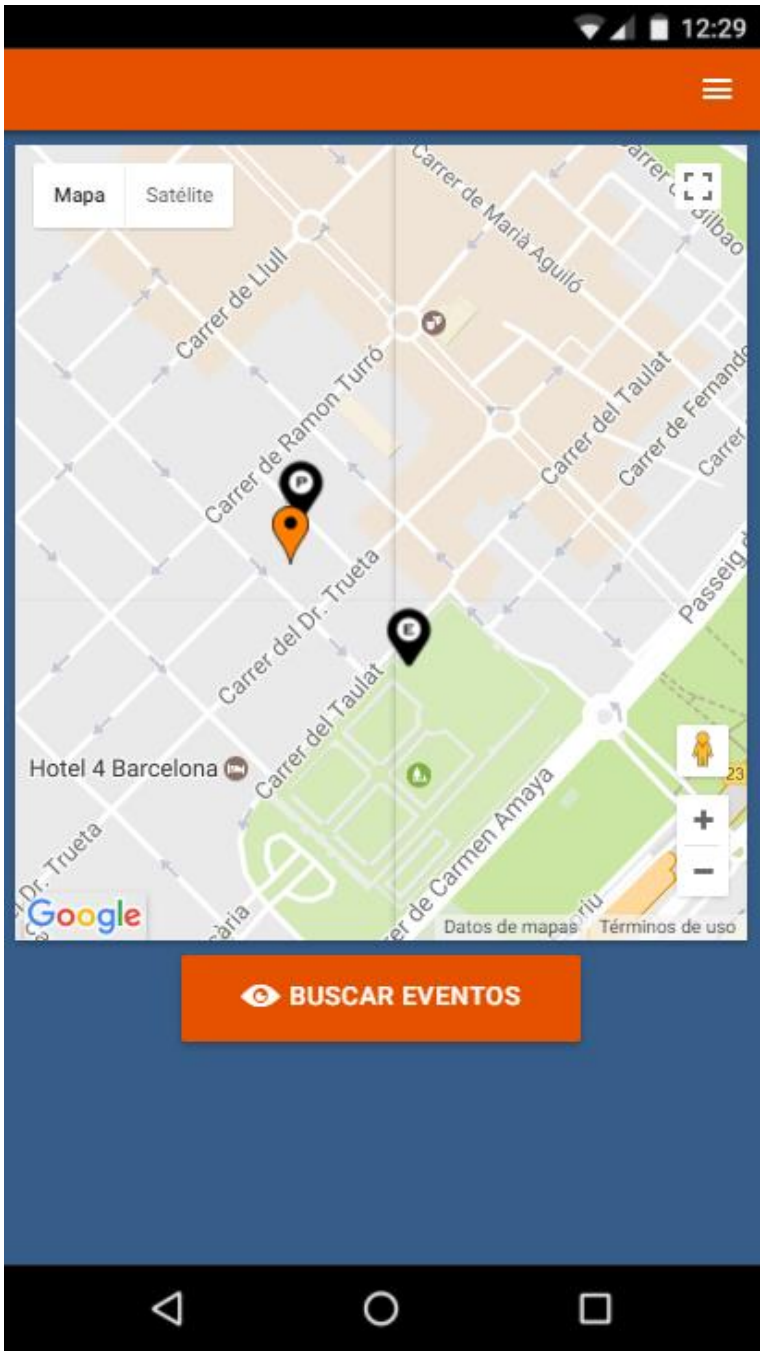


Figura 10. Pantalla "home".

A la figura 10 tenim la nostre Home. El marcador Taronja, ens marca la nostre posició actual, la qual s'actualitza cada cert temps, mentre que despres tenim uns altres 2 marcadors més foscus.

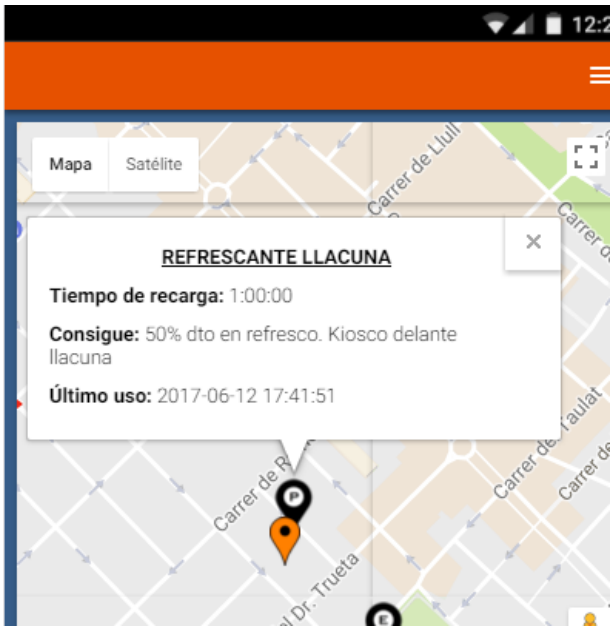


Figura 11. Informació rellevant d'un esdeveniment Parada.

Els marcadors amb la "P", figura 11, indican que son parades, només amb un click podràs aconseguir una recompensa i es carregarà amb el temps. Ens informa del temps de recàrrega, la recompensa que s'obté i quan ha sigut l'últim cop que s'ha utilitzat si encara s'està carregant.

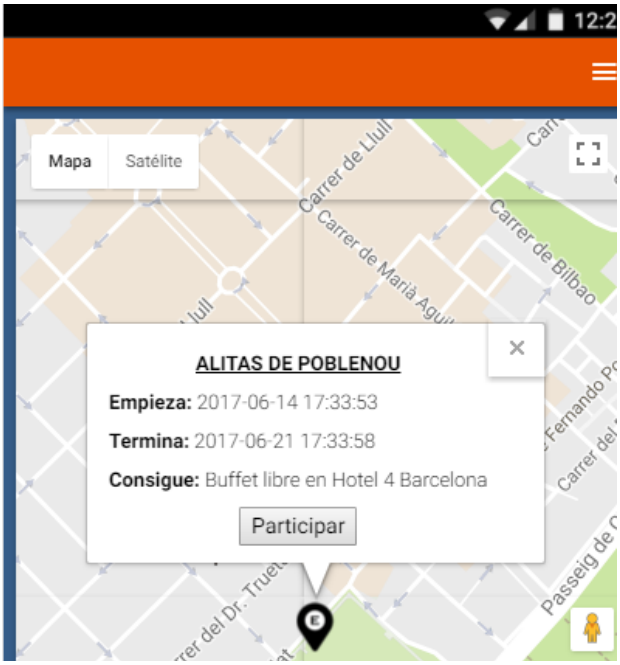


Figura 12. Informació rellevant d'un esdeveniment limitat.

Per un altre banda tenim els esdeveniments limitats, figura 12. Els quals son indicats pels marcadors amb la “E”. Ens informa sobre les dates d’inici i fi i sobre la recompensa que es pot guanyar. També tenim un botó el qual ens enviarà als detalls de l’esdeveniment, el qual veurem més tard.

Lista:

Figura 13. Pantalla llista, amb tots els esdeveniments disponibles del moment.

La figura 13 mostra un llistat de tots els esdeveniments limitats disponibles pel moment. Si un esdeveniment ja no es pot participar, no apareixerà aquí. Aquesta pàgina és la que en un principi es pensava col·locar a la home, encara que fa una millor funció on és ara mateix.

Desde aquí, podem entrar als detalls de qualsevol esdeveniment pero veure la seva descripció llarga, la qual ens ajudarà a poder realitzar-lo i completar l'esdeveniment.

També disposem d'un filtre, que actualment és de provincies, el qual pot ajudar força si vols buscar qué fer un cap de setmana en concret que marxes fora, per exemple.

Evento:

Figura 14. Pantalla "Evento" amb els detalls d'un esdeveniment.

Si fem click a ENTRAR, desde la figura 13, a qualsevol esdeveniment, ens trobarem una pantalla com aquesta, figura 14. Com hem dit anteriorment, aquí trobarem tots els detalls de l'esdeveniment, la direcció exacte, la recompensa i sobre tot com guanyar-la. El botó "PARTICIPAR" conecta amb la càmera del dispositiu i et permet fer una fotografia, que si creus que és correcte, s'enviarà al servidor. Un cop es completi l'esdeveniment, no podràs tornar a accedir-hi.

Un cop tenim el funcionament bàsic, mostrarem el menú, el qual ens permetrà moure'ns per l'aplicació,figura 15. S'obre fent click a l'icona superior dreta.

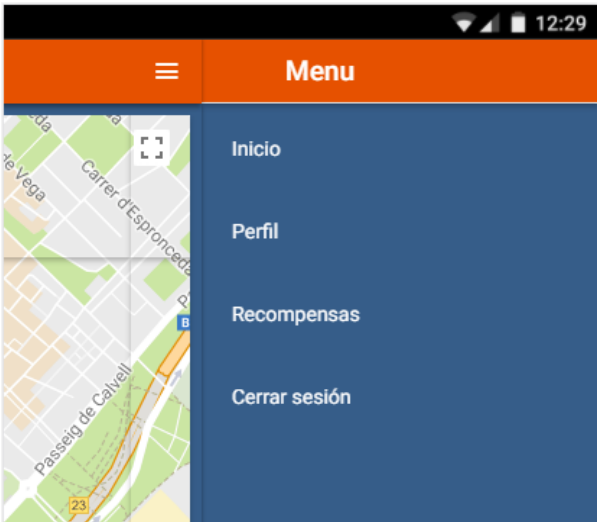


Figura 15. Menú lateral.

Inici:

Ens enviarà de nou cap a la Home.

Perfil:

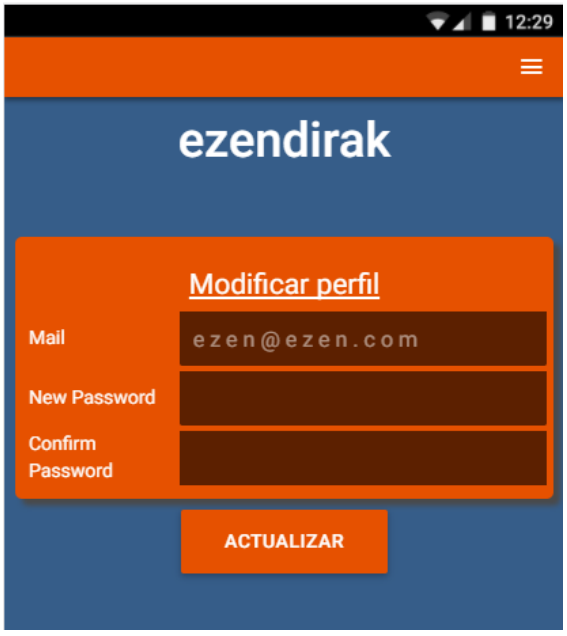


Figura 16. Pantalla "Perfil". Amb les dades modificables del perfil.

A la figura 16, podrem modificar la poca informació que tenim sobre l'usuari. El correu ha de seguir un petit pattern bàsic i la contrasenya, en cas de canviar-la, ha de ser correcte en els 2 labels. Es poden realitzar els canvis dels dos atributs simultàneament. Hi ha un sistema d'alertes preparat pel tema dels possibles errors.

Recompensas:

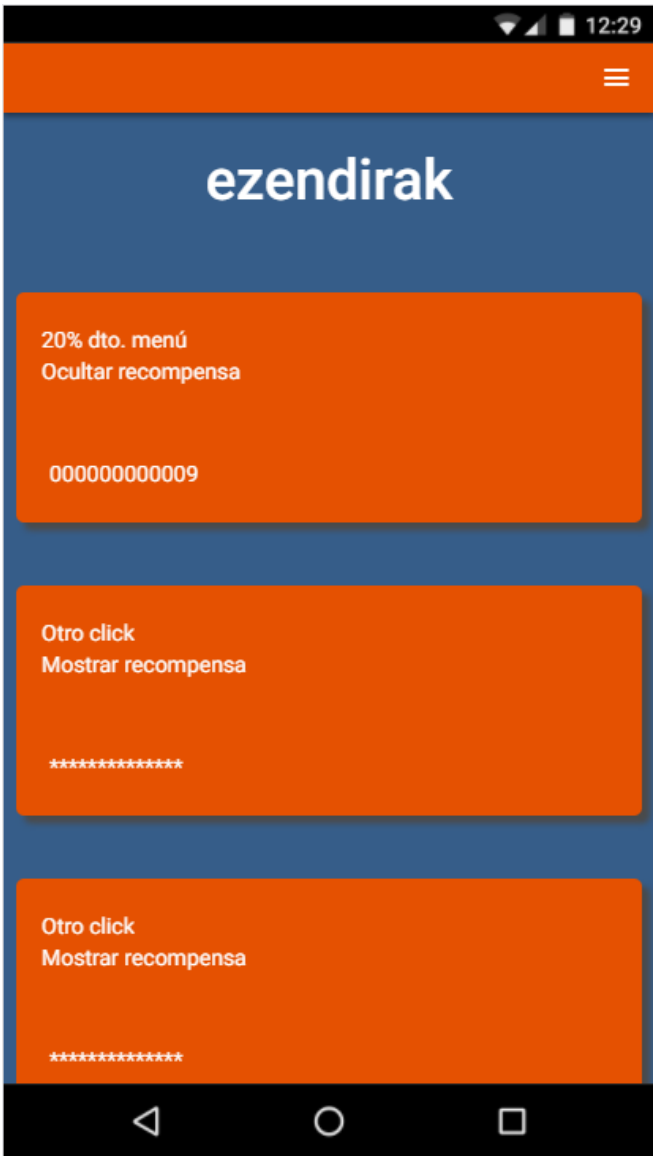


Figura 17. Pantalla "Recompensas". Amb tots els codis adquirits per l'usuari.

A la figura 17 trobarem totes les recompenses adquirides amb el nostre usuari. Ens indica quina recompensa és i el seu codi corresponent. Fent click a Mostrar recompensa es mostra el codi, sino per defecte està amagada. Les recompensa, al igual que els esdeveniments llistats a lista, apareixen amb unes animacions a la pantalla.

Cerrar sesión:

Finalment tenim l'opció de tancar la sessió, logout. Tanca la sessió, elimina el token i ens envia a la pantalla de login.

5. Implementació

En aquesta secció de la memòria detallaré la implementació de la pròpia aplicació i de l'administrador del seu servidor. A disseny gràfic no s'ha parlat del servidor pel fet de que la seva interfície es per defecte i no s'ha realitzat cap canvi sobre ella, per tant, s'explicarà directament desde el punt funcional. Aixi com una petita guia per saber com s'han programat els controladors i amb quines funcions. També entrarem en detall de la diferencia entre realitzar un build per Android, a realitzar-lo per IOs.

a. Administració Servidor Django

Com hem dit anteriorment, en el fitxer admin.py es programa la lógica de les accions realitzables desde l'administració de Django, figura 18.

```
class ModificarEventoLimitado(admin.ModelAdmin):
    list_display = ('event','ini_date','exp_date')
    fieldsets = [
        (None, {'fields': ['event']}),
        (None, {'fields': ['ini_date'],
                'description': "<h5>Date Format: YYYY-M-D<br/>Time Format: H:M:S</h5>"}),
        (None, {'fields': ['exp_date'],
                'description': "<h5>Date Format: YYYY-M-D<br/>Time Format: H:M:S</h5>"}),
        ]
```

Figura 18. Lògica de modificar Esdeveniment Limitat. Els camps dels que consta i el seu format de dades

Per altre banda, a models.py, figura 19, que és on guardem les estructures de la base de dades, també podem trobar totes les funcions returns de les taules i el seu "verbose_name" que serà el nom pel qual els trobarem a l'administrador de Django.

El class Meta ens indica "l'enunciat" que tindrà dins de l'administració de Django al utilitzar aquesta taula. Com que EventoLimitado, depen "d'evento", podem accedir a totes les dades d'aquest registre "d'evento"

```
class EventoLimitado(models.Model):
    event = models.OneToOneField(Evento,on_delete=models.CASCADE,unique=True,verbose_name='Evento')
    ini_date = models.DateTimeField(validators=[fechaIniMayorActual],verbose_name='Fecha de inicio')
    exp_date = models.DateTimeField(validators=[fechaExpMayorIni],verbose_name='Fecha de fin')

    class Meta:
        verbose_name_plural = "Convertir evento en evento limitado"
```

Figura 19. Taula EventoLimitado, vist des de models.py

```
def __str__(self):
    return '['+str(self.event.id)+'] '+self.event.brand+': '+self.event.event_name

def returnJSON(self):
    ini_date_formated = str(self.ini_date).split("+")[0]
    exp_date_formated = str(self.exp_date).split("+")[0]
    return {'id':str(self.event.id),'brand':self.event.brand,'event_name':self.event.event_name,'event_d

def returnMap(self):
    ini_date_formated = str(self.ini_date).split("+")[0]
    exp_date_formated = str(self.exp_date).split("+")[0]
    return {'id':str(self.event.id),'event_name':self.event.event_name.upper(),'event_type':'limitado','

def isAvaible(self):
    ini_date_formated = str(self.ini_date).split("+")[0]
    exp_date_formated = str(self.exp_date).split("+")[0]
    date_today_formated = str(datetime.today()).split(".")[0]
    if (date_today_formated > ini_date_formated) and (date_today_formated < exp_date_formated):
        return True
    return False

def getPosition(self):
    return {'latitud':self.event.latitud,'longitud':self.event.longitud}
```

Figura 20. Diferents Returns de la taula EventoLimitado, vist des de models.py

Com es pot veure a la figura 20, tenim diferents funcions per a la mateixa taula. Amb Django, es realitzen diferents returns, si és necessari, per diferents necessitats de l'aplicació. En aquest cas, per a la taula EventoLimitado, que ens indica els esdeveniments limitats, tenim 4 returns diferents:

- returnJSON: Aquest return el tenim a totes les taules. És el return bàsic que ens retorna tota la info del registre en format json.
- returnMap: És la funció que passa els esdeveniments limitats disponibles a l'api de maps, que més tard els mostra al mapa.
- isAvaible: Ens indica si l'esdeveniment ha passat o encara no.
- getPosition: Ens proporciona la geolocalització de l'esdeveniment per poder-lo carregar dins del maps.

Un cop explicada l'estructura interna, anem a mostrar com es l'administració desde dins, com crear diferents esdeveniments i configurar-los.

En aquest projecte, utilitzem la consola Winpython per executar el servidor Django.

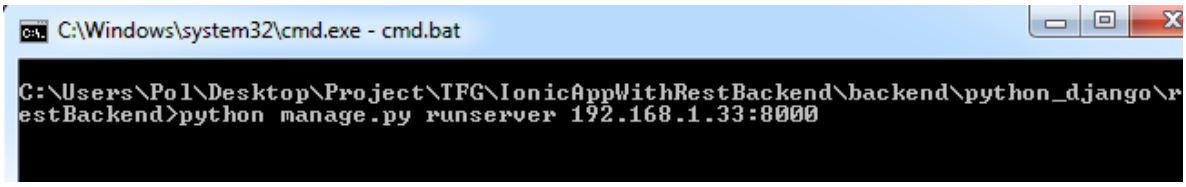


Figura 21. Consola de WinPython. Preparada per executar el servidor.

Com es veu a la figura 21, s'executa en local amb una ip i un port preestablerts.
Un cop cridem al server, des de la consola podem comprovar les urls a les que accedeix el servidor, figura 22, i si entrem per local a la url que hem iniciat el servidor afegint /admin, figura 23.

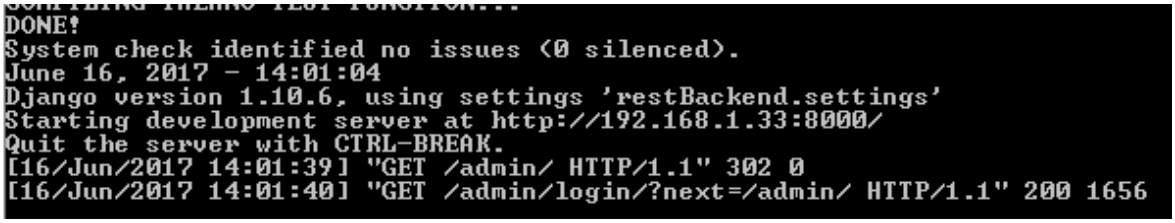


Figura 22. Consola de WinPython. Servidor executat.

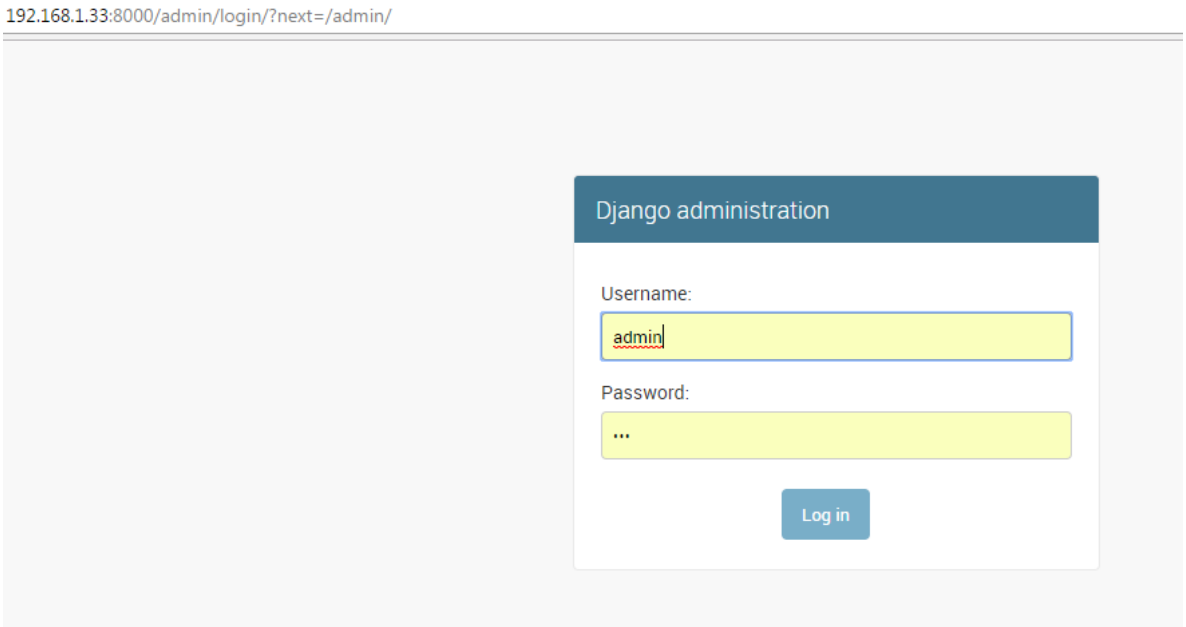


Figura 23. Pàgina de login de l'administrador de python. Des de navegador chrome.

Necessitem un compte d'administrador per poder entrar, i ens redirigirà a la següent pantalla, figura 24.

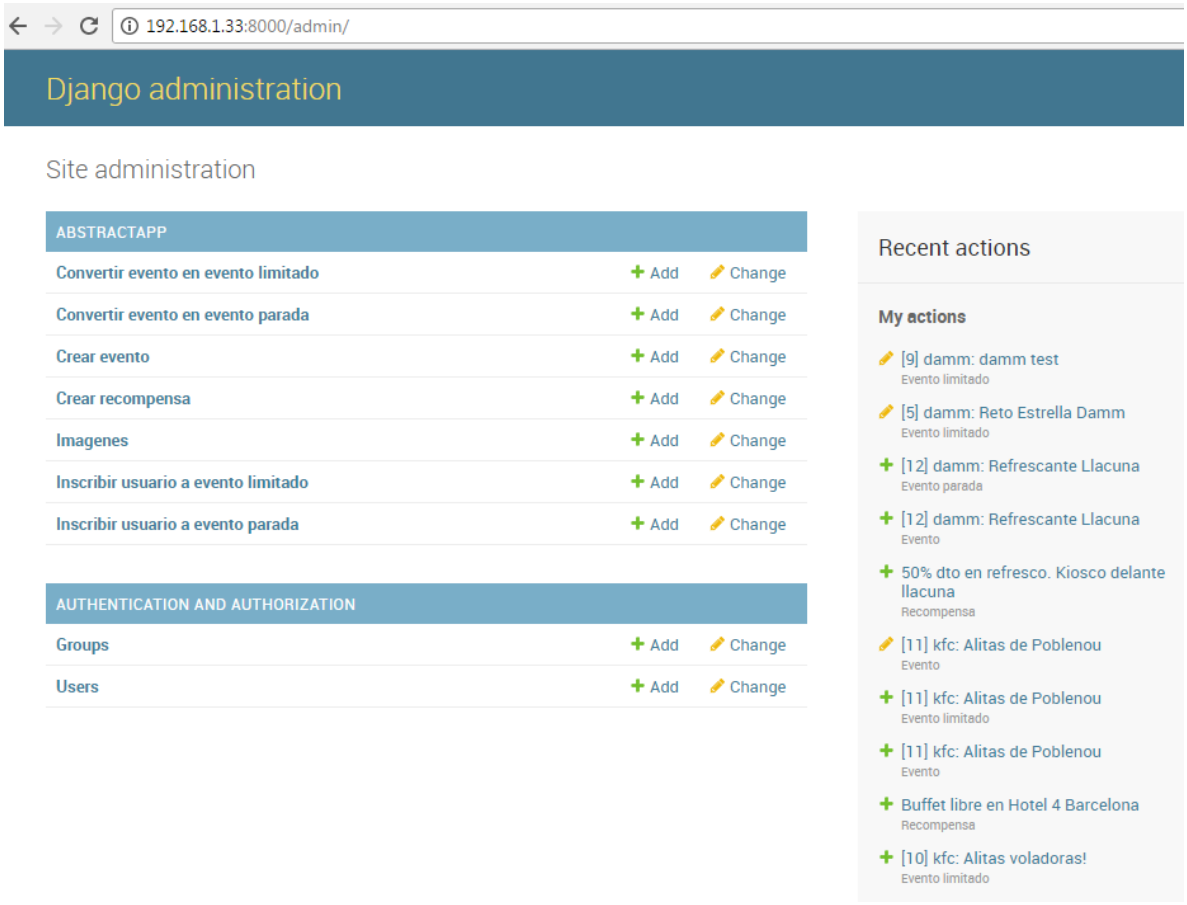


Figura 24. Pàgina inicial de l'administrador.

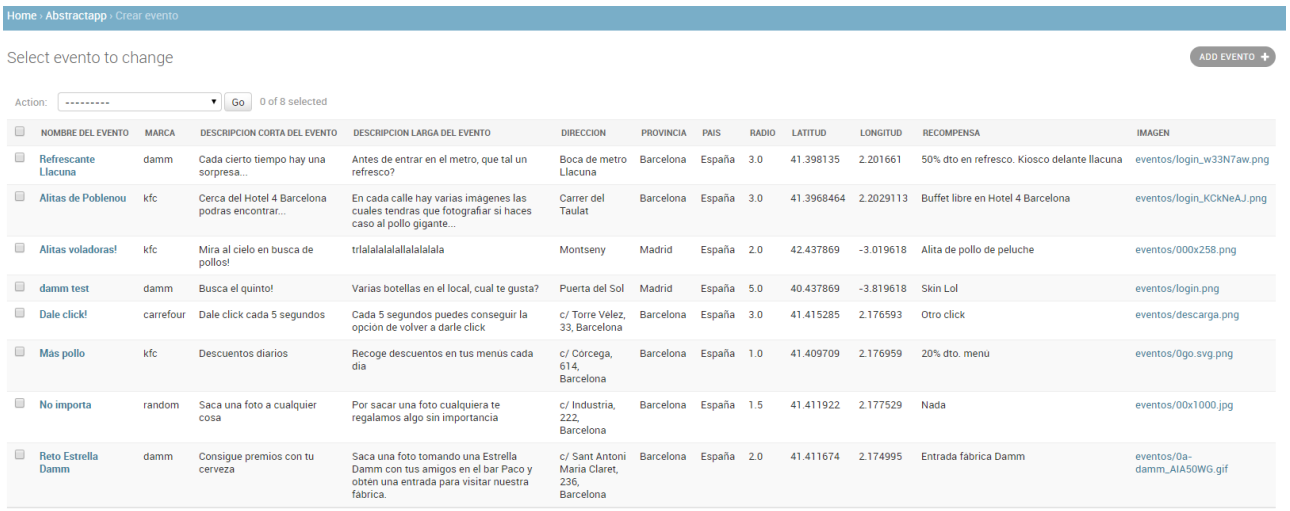
Com es pot veure a la figura 24, des d'aqui es pot crear qualsevol tipus d'esdeveniment i d'usuari.

Pel que fa la seva estructura, com hem dit anteriorment, s'han creat les taules d'abstractApp i s'ha utilitzat la de Users que ens proporcionen per defecte. La de Groups no s'ha utilitzat en cap moment.

Crear esdeveniment

Desde abstractApp, podem crear qualsevol nou esdeveniment. Per ensenyar el seu funcionament realitzarem les pases per crear un nou esdeveniment limitat.

Inicialment farem click a Crear evento i apareixerà un llistat amb tots els esdeveniments actuals i les seves dades, figura 25.



NOMBRE DEL EVENTO	MARCA	DESCRIPCION CORTA DEL EVENTO	DESCRIPCION LARGA DEL EVENTO	DIRECCION	PROVINCIA	PAIS	RADIO	LATITUD	LONGITUD	RECOMPENSA	IMAGEN
Refrescante Llacuna	damm	Cada cierto tiempo hay una sorpresa...	Antes de entrar en el metro, que tal un refresco?	Boca de metro Llacuna	Barcelona	España	3.0	41.398135	2.201661	50% dto en refresco. Kiosco delante llacuna	eventos/login_w33N7aw.png
Alitas de Poblenou	kfc	Cerca del Hotel 4 Barcelona podras encontrar...	En cada calle hay varias imagenes las cuales tendras que fotografiar si haces caso al pollo gigante...	Carrer del Taulat	Barcelona	España	3.0	41.3968464	2.2029113	Buffet libre en Hotel 4 Barcelona	eventos/login_KCkNeAJ.png
Alitas voladoras!	kfc	Mira al cielo en busca de pollos!	trlalalalalallalalalala	Montseny	Madrid	España	2.0	42.437869	-3.019618	Alita de pollo de peluche	eventos/000x258.png
damm test	damm	Busca el quinto!	Varias botellas en el local, cual te gusta?	Puerta del Sol	Madrid	España	5.0	40.437869	-3.819618	Skin Lol	eventos/login.png
Dale click!	carrefour	Dale click cada 5 segundos	Cada 5 segundos puedes conseguir la opcion de volver a darle click	c/ Torre Velez, 33, Barcelona	Barcelona	España	3.0	41.415285	2.176593	Otro click	eventos/descarga.png
Más pollo	kfc	Descuentos diarios	Recoge descuentos en tus menús cada dia	c/ Córcega, 614, Barcelona	Barcelona	España	1.0	41.409709	2.176959	20% dto. menu	eventos/0go.svg.png
No importa	random	Saca una foto a cualquier cosa	Por sacar una foto cualquiera te regalamos algo sin importancia	c/ Industria, 222, Barcelona	Barcelona	España	1.5	41.411922	2.177529	Nada	eventos/00x1000.jpg
Reto Estrella Damm	damm	Consigue premios con tu cerveza	Saca una foto tomando una Estrella Damm con tus amigos en el bar Paco y obtén una entrada para visitar nuestra fábrica.	c/ Sant Antoni Maria Claret, 236, Barcelona	Barcelona	España	2.0	41.411674	2.174995	Entrada fábrica Damm	eventos/0a-damm_AIA50WG.gif

Figura 25. Llistat dels dos tipus d'esdeveniment actuals.

Fent click a add evento a la part superior dreta ens enviarà al formulari el qual ens demanarà les dades d'un esdeveniment bàsic, figura 26.

Figura 26. Formulari per afegir un nou esdeveniment.

La marca ens indica a quina classe atacarà del model d'imatges, des d'un esdeveniment limitat, al realitzar la fotografia requerida.

Les coordenades de latitud i longitud s'agafen des de google maps. Es requereix una imatge per crear l'esdeveniment i poder-la mostrar en els detalls de l'esdeveniment (vista "evento"), encara que en aquesta versió de l'aplicació no s'utilitza ja que no és necessari pel testing. Aixi mateix, l'atribut radio està pensat per que controli la distància permesa per participar a l'esdeveniment. Com que aquest projecte funciona únicament en local, aquesta funció està desactivada, però si que s'utilitzen les coordenades per mostrar només els esdeveniments a 1km de distància, aproximat.

En aquest moment, aquest esdeveniment no és ni limitat ni parada, però encara així necessita una recompensa. Si no tenim cap recompensa disponible, entenent disponible com una recompensa que no s'utilitza en cap altre esdeveniment, podem crear-la aquí mateix, fent click al símbol "+" de color verd, situat a la dreta de recompensa, figura 23.

Ens apareixerà una finestra emergent per crear la nova recompensa. Simplement demanarà un nom, figura 27.

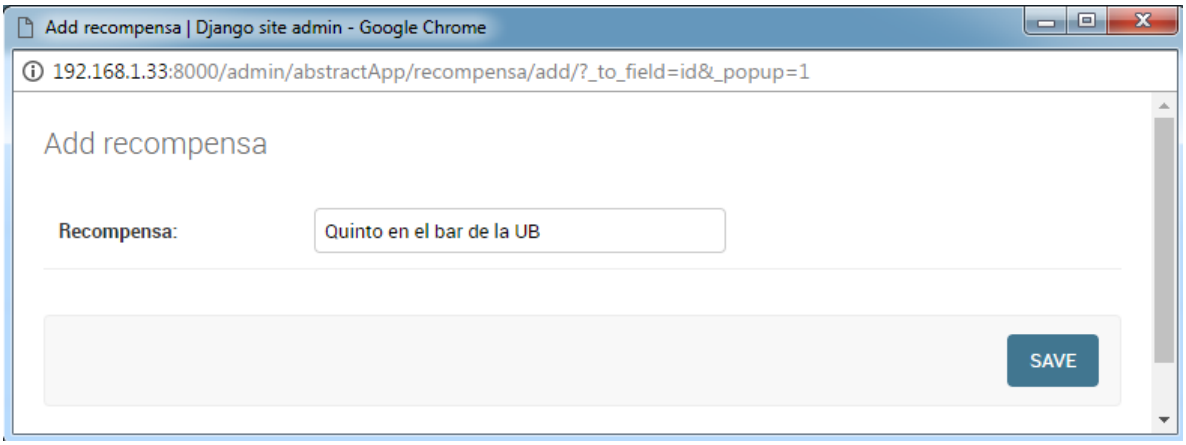


Figura 27. Creació de recompensa.

Un cop tenim tota la informació de l'esdeveniment, guardem fent click a "save" abaix a la dreta, i el nou esdeveniment ens apareixerà a la llista d'esdeveniments, figura 28.

Figura 28. Llistat dels esdeveniments amb el nou esdeveniment.

En aquest moment hem de decidir si l'esdeveniment serà limitat o parada. Per realitzar aquest exemple, el farem limitat, però també es mostrarà la pantalla del parada.

Des de la figura 24, accedim a "Convertir evento en evento limitado" i se'ns llistaran tots els esdeveniments limitats, amb la possibilitat d'afegir-ne de nous fent click a "Add evento limitado",situat a dalt a la dreta, figura 29.

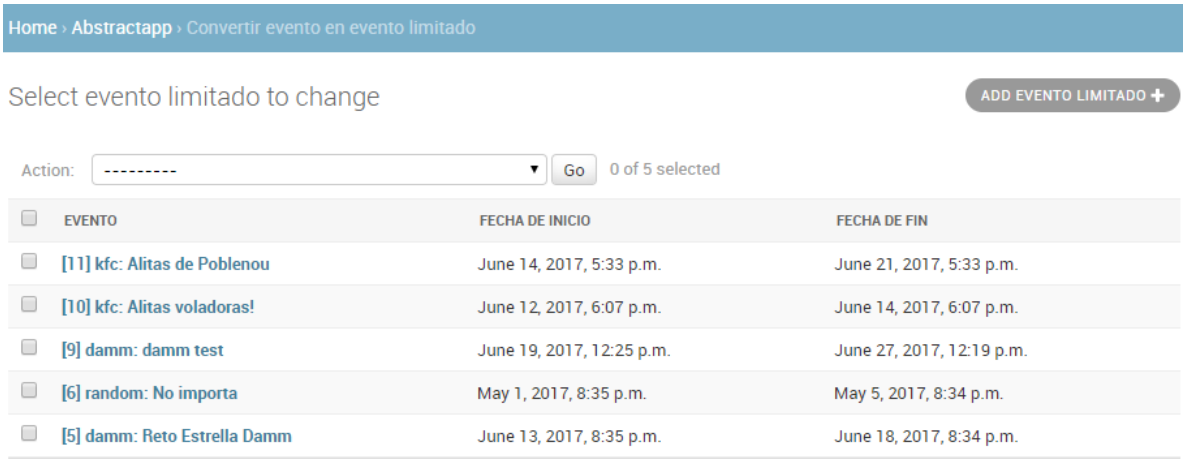


Figura 29. Llistat dels esdeveniments limitats.

En el cas d'un esdeveniment limitat, necessitarem les dates d'inici i fi, figura 30.

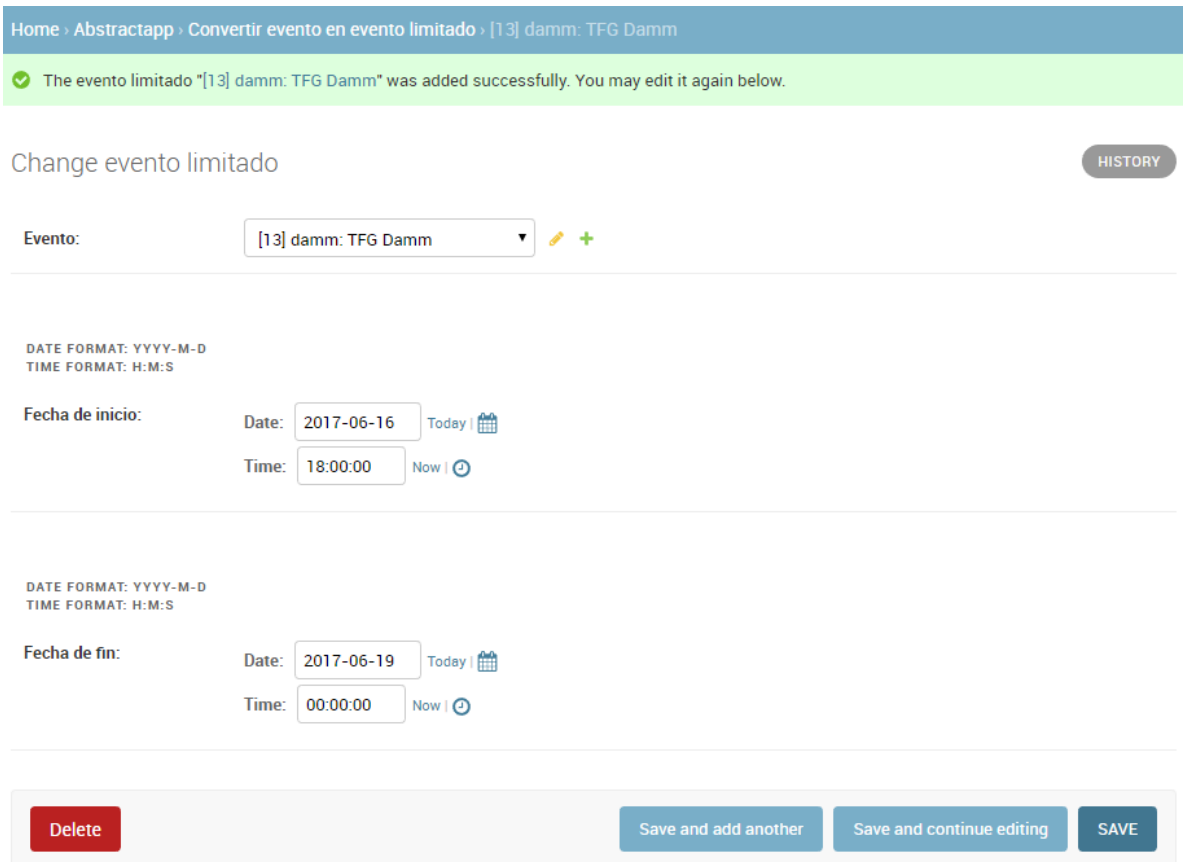


Figura 30. Afegir/Modificar esdeveniment limitats.

Mentre que en el cas dels esdeveniments parada, només necessitarem el temps de recàrrega, figura 31.

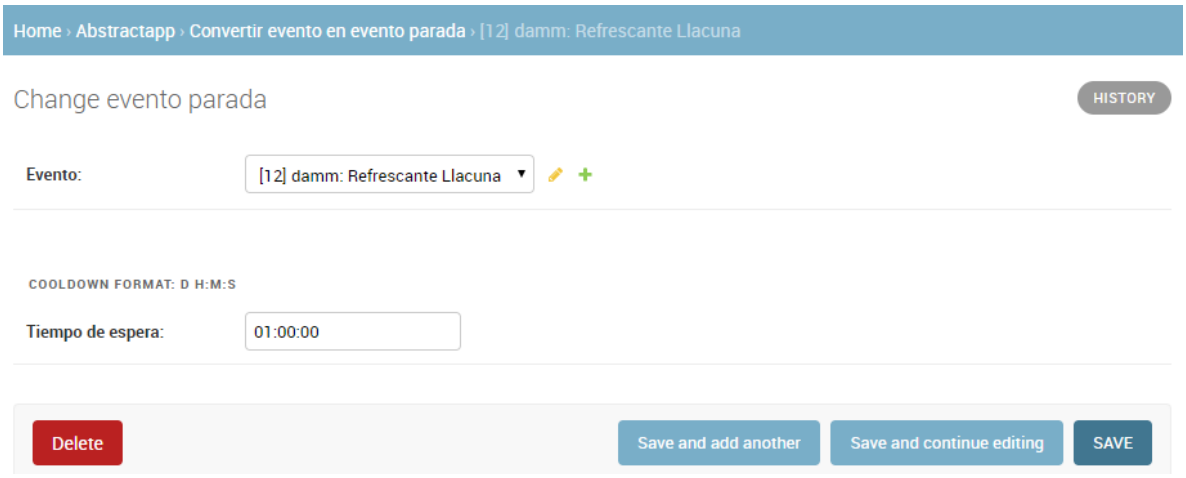


Figura 31. Afegir/Modificar esdeveniment parada.

Les altres 3 opcions d'AbstractApp, estan més pensades per el control d'usuaris.

- Imagenes: Podem trobar totes les imatges enviades des d'esdeveniments limitats, figura 32.

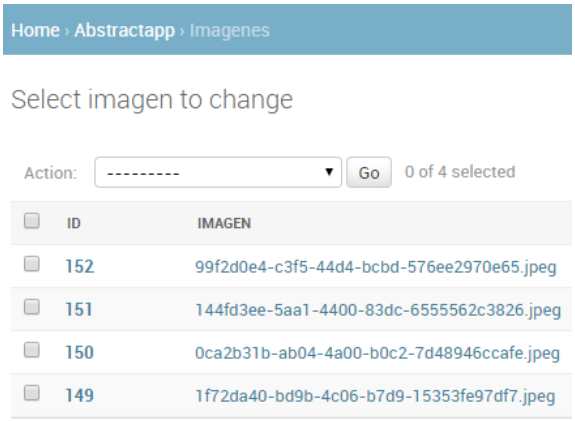


Figura 32. Imatges rebudes d'esdeveniments limitats.

- Inscribir usuario a evento limitado: Ens indica quins usuaris han participat en quins esdeveniments limitats, figura 33.

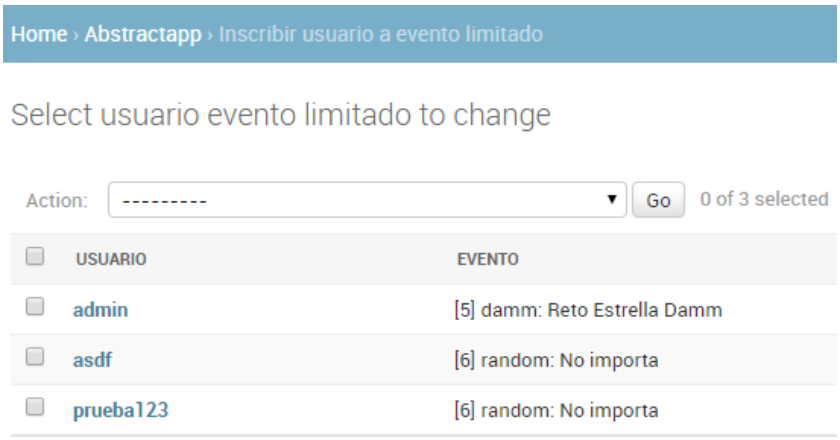


Figura 33. Usuaris i les seves participacions a esdeveniments limitats.

- Inscribir usuario a evento parada: Ens indica quins usuaris han participat en quins esdeveniments parades, indicant el seu últim ús de la parada, figura 34.

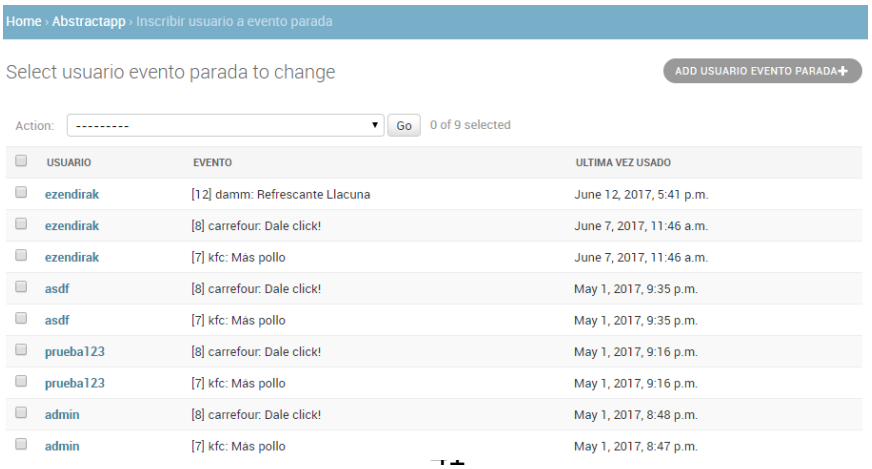


Figura 34. Usuaris i les seves participacions a esdeveniments parada.

b. Aplicació Ionic

Controladors

En aquest projecte tenim un controlador per vista, i ademés, alguns més per les funcionalitats del maps, del tractament d'imatges per l'enviament i per un par més de funcions que són les de la creació del Token i pel localstorage, el qual ens permetrà mantenir informació de la sessió dins del telèfon, com per exemple el nom d'usuari.

El fitxer inicial es "app.js". Aqui carreguem tots els mòduls que utilitzarem en l'aplicació. En aquest projecte utilitzem com a extres 'ionic-material'(textures, animacions) i 'ionMdInput'(nous components, efectes), figura 35.

Un cop tenim els mòduls, configurem la url i es relaciona cada estat de l'aplicació a una vista i un controlador específic. Aquest projecte s'ha realitzat en connexió local, ethernet/wifi, en tot moment, ja que no disposem de servidors amb ip pública fixe, per això la nostre Url és: "192.168.1.33:8000".

```
angular.module('myIonicApp', ['ionic', 'myIonicApp.controllers', 'ngCordova', 'ionic-material', 'ionMdInput'])

.constant('ApiEndpoint',{
  url: 'http://192.168.1.33:8000/'
})
.run(function($ionicPlatform) {
  $ionicPlatform.ready(function() {
    if (window.cordova && window.cordova.plugins.Keyboard) {
      cordova.plugins.Keyboard.hideKeyboardAccessoryBar(true);
      cordova.plugins.Keyboard.disableScroll(true);

      $httpProvider.defaults.useXDomain=true;
      delete $httpProvider.defaults.headers.common['X-Requested-With'];
    }
    if (window.StatusBar) {
      StatusBar.styleDefault();
    }
  });
})

.config(function($stateProvider, $urlRouterProvider) {
  $stateProvider

  .state('app', {
    url: '/app',
    abstract: true,
    templateUrl: 'templates/menu.html',
    controller: 'AppCtrl'
  })
  .state('app.login', {
    cache: false,
    url: '/login',
    views: {
      'menuContent': {
          templateUrl: 'templates/login.html',
          controller: 'LoginCtrl'
      }
    }
  })
```

Figura 35. Routing de les vistes.

Declarem el controlador amb un nom únic i li passem les variables que necessiti, figura 36. Algunes de les que utilitzem són les següents:

- $scope: informació que s'intercanvia entre el controlador i la vista.
- $http: per l'enviament de dades.
- ApiEndpoint: url per consultar el servidor.
- Token: Serà requerit pràcticament a qualsevol controlador.
- $ionicPopup: S'utilitza quan requerim de l'aparició d'un Popup.
- $cordovaGeolocation: Es necessita el pas d'aquesta variable en el moment que s'utilitzi el plugin de GEO localització.
- $stateParams: Ens indica que utilitzarem variables passades per paràmetre (per url).

```
var module = angular.module('myIonicApp.controllers');

module.controller('EventCtrl',function($scope,$http,$ionicPopup,ApiEndpoint,$cordovaGeolocation,$stateParams, Token){
```

Figura 36. Declaració del controlador. Pas de paràmetres.

A continuació es comprova que el token sigui correcte, i es realitza la funció requerida, que pot ser una consulta a la base de dades o omplir un select, per exemple.

L'estructura de tots els controladors es força semblant així que no els explicaré tots, però sí que entrarem en detalls sobre els controladors més importants. Aquests son:

- Esdeveniments (eventCtrl): S'encarrega dels detalls dels esdeveniments.
- Llistat (listCtrl): Ens llista tots els esdeveniments limitats disponibles.
- Maps (mapCtrl): Mostra la nostre posició i els esdeveniments propers.
- Perfil (perfilCtrl): És l'únic amb el qual l'usuari pot modificar dades a la base de dades.

eventCtrl

```
var module = angular.module('myIonicApp.controllers');

module.controller('EventCtrl',function($scope,$http,$ionicPopup,ApiEndpoint,$cordovaGeolocation,$stateParams, Token){

  $scope.$on('$ionicView.enter', function() {
    Token.isToken()

  })

  $scope.header = "Informacion evento";
  $scope.evento = [];
  $http({
    method: 'GET',
    url: ApiEndpoint.url+ 'evento/' + $stateParams.id +'/',
  }).then(function successCallback(response) {
      $scope.evento = [];
      for(var r in response.data) {
        var info = response.data[r];
        $scope.evento.push(info);
      }

  }, function errorCallback(response) {
      console.log("ERROR");
  });
})
```

Figura 37. Estructura d'un controlador.

Per realitzar una consulta per recuperar certa informació es pot realitzar de 2 formes, POST i GET.

La gran part de les que s'han realitzat son POST, pel fet de que no s'envien les dades per la url, per tant no s'altera la seva estructura i podem tenir la informació més organitzada. En el cas del controlador d'esdeveniments, utilitzem un GET, figura 37. Ja que així podem consultar els esdeveniments individualment des de la seva Id.

Aquesta petició s'envia cap a views.py, directament al model indicat en la Url de la petició js, on es realitzarà la consulta, figura 38. En aquest cas realitzem una consulta a evento/+$stateParams.id que es tradueix en el model evento que requereix d'una Id.

```
def evento(request,id):

    evento = EventoLimitado.objects.get(event_id=id)
    prepareToSend = []
    prepareToSend.append(evento.returnJSON())
    datos = json.dumps(prepareToSend)

    return HttpResponse(datos)
```

Figura 38. Model que ens retorna els detalls d'un esdeveniment concret.

Des del servidor podem comprovar quina informació estem enviant de volta al js d'Ionic. En el cas "d'eventos", al entrar a l'esdeveniment anomenat Reto Estrella Damm, ens apareix un return amb la següent estructura, figura 39.

```
[{"event_type": "limitado", "brand": "damm", "reward": "Entrada f\u00e1brica Dam
m", "event_direccion": "c/ Sant Antoni Maria Claret, 236, Barcelona", "ini_date"
: "2017-06-13 20:35:04", "event_name": "Reto Estrella Damm", "event_fullDescript
ion": "Saca una foto tomando una Estrella Damm con tus amigos en el bar Paco y o
bt\u00e9n una entrada para visitar nuestra f\u00e1brica.", "event_description":
"Consigue premios con tu cerveza", "id": "5", "latitud": "41.411674", "exp_date"
: "2017-06-18 20:34:13", "longitud": "2.174995"}]
[17/Jun/2017 13:15:12] "GET /evento/5/ HTTP/1.1" 200 529
```


Figura 39. Resposta en format json sobre els detalls d'un esdeveniment.

En aquest moment, tenim la funció de succes Callback, la qual recull la informació que ens arriba desde Django i l'afegeix a una variable $scope anomenada evento. Aquesta variable evento, és amb la que treballarem de cara al frontend d'Ionic, figura 40.

```
<ion-view class="appBackground">
  <ion-content>
    <div class="card sombres eventColor whiteFont">
      <div class="item eventTitle whiteFont">
        Detalles de {{evento[0].event_name}}
      </div>
      <div>
        <div class = "row">
          <div class = "col subrayado">Inicio</div>
        </div>
          <div class = "row">
            <div class = "col ">{{evento[0].ini_date}}</div>
          </div>
          <div class="row">
            <div class = "col subrayado">Descripcion del Evento</div>
          </div>
          <div class="row">
            <div class = "col">{{evento[0].event_fullDescription}}</div>
          </div>

          <div class="row">
            <div class = "col subrayado">Recompensa</div>
          </div>

          <div class="row">

            <div class = "col">{{evento[0].reward}}</div>
          </div>
          <div class = "row">
            <div class = "col"><div class="subrayado">Lugar </div>{{evento[0].event_direccion}}</div>

          </div>
          </div>
    </div>
    <div style="text-align: center;">

      <button class="button button-test2" ng-controller="ImageCtrl" ng-click="addMedia(evento[0].brand,evento[0].id)"><i class="icon ion-camera
      "></i> Participar</button>
    </div>
  </ion-content>
</ion-view>
```

Figura 40. Estructura de la vista "evento" d'Ionic.

Des d'el frontend d'Ionic podem treballar amb la variable com si fos un objecte, tenint tota la informació del json.

Totes les consultes s'han testejat amb tots els tipus de dades que es poden utilitzar a cada camp. En el cas de que hi hagués algun error, s'informaria des del servidor. En la part Ionic, s'han controlat els errors en els cassos de modificació d'informació de l'usuari, dels que parlarem més endavant.

listCtrl:

Ara veurem el funcionament del controlador de la vista “lista”. S’encarrega de 3 funcions:

- Mostrar tots els esdeveniments
- Mostrar els esdeveniments filtrats
- Omplir filtres disponibles

Totes les consultes que realitza, son mitjançant POST. Com a tots els controladors, inicialment es comprova el token i després es realitzen les funcions.

La principal és la de llistar tots els esdeveniments disponibles, figura 41.

```
$http({
  method: 'POST',
  url: ApiEndpoint.url+ 'lista/',
  data: $localstorage.get('name')
}).then(function successCallback(response) {
    $scope.lista = [];
    var ahora = new Date()
    for(var r in response.data) {
      var evento = response.data[r];
      fechaE = new Date(evento.exp_date.slice(0,10));
      if (fechaE > ahora){
        $scope.lista.push(evento);
      }
    };

}, function errorCallback(response) {
    console.log("ERROR Lista");
});
```

Figura 41. Consulta per llistar tots els esdeveniments.

Podem observar com realitzem un POST, a la url anomenada lista/ i passant un data amb el nom de l’usuari. Amb aquest nom aconseguim l’Id per mostrar els esdeveniments on no ha participat l’usuari i ens estalviem enviar la Id.

A la figura 42, podem veure el funcionament del model “lista”.

```python
@csrf_exempt
def lista(request):
    user = str(request.body,"UTF-8")
    try:
        user = User.objects.get(username=user)
        eventos = EventoLimitado.objects.all()
        prepareToSend = []
        for evento in eventos:
            if not UsuarioEventoLimitado.objects.filter(event_id=evento.id,user_id=user.id):
                prepareToSend.append(evento.returnJSON())
        print(prepareToSend)
        datos = json.dumps(prepareToSend)
    except:
        datos = "ERROR Lista"
    return HttpResponse(datos)
```

Figura 42. Model que ens retorna els esdeveniments limitats.

Comprovem els esdeveniments que tenim i, amb el nom d l'usuari aconseguim el seu Id i comprovem que aquest usuari no hagi participat en cap esdeveniment que anem a mostrar. Un cop els tenim, fem un return en format JSON cap el js un altre cop.

La funció de Callback rep tots els esdeveniments, i s'encarrega de filtrar al moment els esdeveniments disponibles. Aquesta opció es podria realitzar des de la consulta al servidor, però al final s'ha programat al js per un motiu.

És el primer cop que programo amb aquest llenguatge i he volgut provar el "potencial" que pot tenir i si, comparat amb realitzar la consulta filtrada directament, hi ha molta diferència en el temps de càrrega, i la resposta és no. Funciona a la mateixa velocitat però el càlcul es realitza des de el telèfon. En el cas de que el servidor funcionés molt lent, sempre seria més precís el filtre des de el telèfon, un cop tenim tots els esdeveniments.

A continuació veurem un altre funció d'aquest controlador: Omplir els filtres disponibles.

Per aquesta versió del projecte només està disponible el filtre de províncies, ja que el de dates donava problemes en IOs, ja que s'ha d'utilitzar un plugin extern i les eines natives no ens serveixen. Omplim el filtre de províncies mitjançant aquesta consulta, figura 43.

```
$http({
    method: 'POST',
    url: ApiEndpoint.url+ 'provincias/',
    data: {'name' : $localstorage.get('name')}
  }).then(function successCallback(response) {
      $scope.listOfOptions2 = [];
      for(var r in response.data) {

        var provincia = response.data[r];
        $scope.listOfOptions2.push(provincia);
      };
      /*console.log(response.data)*/
      console.log($scope.listOfOptions2)
  }, function errorCallback(response) {
      console.log("ERROR Provincia");
  });
```

Figura 43. Petició en POST per mostrar províncies on es realitzen esdeveniments.

És força senzilla. Fem petició en POST a provincias/ i enviem el nom de l'usuari que està guardat al telèfon. El resultat que rebrem serà redirigit cap a la variable que forma el select a la part Ionic, figura 44.

Figura 44. Resultat del filtre de províncies.

La consulta que es realitza a views.py és simple. Un distinct amb les províncies disponibles, figura 45.

```python
@csrf_exempt
def provincias(request):
    infoUser = json.loads(str(request.body,"UTF-8"))
    if not infoUser or 'name' not in infoUser:
        return HttpResponse('false')
    try:
        provincias = Evento.objects.values_list('event_provincia').distinct()
        prepareToSend = []
        for provincia in provincias:
            prepareToSend.append(provincia)
        datos = json.dumps(prepareToSend)

        return HttpResponse(datos)
    except:
        return HttpResponse('false')
```

Figura 45. Consulta que ens retorna les províncies diferents.

Encara que no l'utiltizem en aquest moment, seguim enviant el nom del localstorage. En tota la sessió l'hem de tenir, per tant, aprofitem per comprovar si el seguim tenint. El resultat que ens retorna, en aquest cas no es en json, és un queryset de django, que el tractarem com un array de strings, figura 46.

```
[["Barcelona"], ["Madrid"]]
[19/Jun/2017 20:52:25] "POST /provincias/ HTTP/1.1" 200 27
```

Figura 46. Resposta vista des del servidor per omplir el filtre de províncies.

Per últim, tenim la petició per a la llista filtrada, la qual està vinculada en el canvi de la select i també inclou el control de les dates pels esdeveniments passats, figura 47. Enviem com a dades: El nom per aconseguir l'Id, el tipos de filtre i l'informació a filtrar.

```
$scope.selectedItemChanged2 = function(selectedItem){
  console.log($scope.listOfOptions2)
  $scope.selectedItem2 = selectedItem
  $scope.lista = []
  tipo = $scope.tipo
  info = $scope.selectedItem2
  $http({
    method: 'POST',
    url: ApiEndpoint.url+ 'listaFiltrada/',
    data: {'username' : $localstorage.get('name'), 'type' : tipo, 'info' : info}
  }).then(function successCallback(response) {
    var ahora = new Date()
      $scope.lista = [];
      for(var r in response.data) {
        var eventoPro = response.data[r];
        fechaE = new Date(eventoPro.exp_date.slice(0,10));
          if (fechaE > ahora){
            $scope.lista.push(eventoPro);
          }
      };
  }, function errorCallback(response) {
      console.log("ERROR Filtro");
  });
}
```

Figura 47. Petició en POST per mostrar els esdeveniments d'una província en concret.

A la figura 48, tenim la consulta que es realitza per a la llista filtrada. Des del select enviem un caràcter el qual ens indica quin filtre utilitzar. Com hem dit anteriorment, en la versió d'aquest projecte només tenim el filtre de provincies, però l'estructura està preparada per tenir-ne més.

Un cop tenim els esdeveniments, ens asegurem de que no siguin parades i l'usuari no hagi guanyat ja la recompensa, i s'envia en format json.

```python
@csrf_exempt
def listaFiltrada(request):

    info = json.loads(str(request.body,"UTF-8"))
    if not info or 'username' not in info or 'type' not in info or 'info' not in info:
        return HttpResponse("ERROR Filtro")
    try:
        if info['type'] == "f":
            # FILTRAR POR FECHA
            eventos = EventoLimitado.objects.all()

        elif info['type'] == "l":
            # FILTRAR POR LOCALIDAD
            prov = info['info'][0]

            eventos = Evento.objects.all().filter(event_provincia=prov)


        user = User.objects.get(username=info['username'])
        prepareToSend = []
        for evento in eventos:
            if EventoLimitado.objects.filter(event_id=evento.id):
                event = EventoLimitado.objects.filter(event_id=evento.id)
                if not UsuarioEventoLimitado.objects.filter(event_id=event[0].event_id,user_id=user.id):
                    prepareToSend.append(event[0].returnJSON())

        datos = json.dumps(prepareToSend)
        return HttpResponse(datos)
    except:
        datos = "ERROR Filtro"

    return HttpResponse(datos)
```

Figura 48. Consulta que ens retorna la llista filtrada.

mapCtrl:

El controlador del maps s'encarrega de totes les funcions relacionades amb la geo-localització i col·locació d'esdeveniments. Per això configurem les variables inicials per poder començar a crear el mapa i totes les marques, figura 49.

```javascript
var infoWindowClose = false;
var latLng = null;
var options = {timeout: 10000, enableHighAccuracy: true};
var myImage = 'img/userMaps.png';
map = new google.maps.Map(document.getElementById("map"));

// My marker
var myMarker = new google.maps.Marker({
        map: map,
        icon: myImage
});

var markers = [];
```

Figura 49. Inicialització del controlador del maps.

La variable "myMarker" serà el marcador que representarà a l'usuari mentre que l'array markers s'omplirà amb els esdeveniments propers. El maps es carregarà gràcies a la API de

google, de la qual tenim una clau que es crida a index.html juntament amb tota la resta d'scripts, figura 50.

```
<script src="cordova.js"></script>
  <script src="bower_components/angular-base64/angular-base64.js"></script>

<script src="http://maps.google.com/maps/api/js?key=KEY_GOOGLE_MAPS_API&sensor=true"></script>

<script src="js/app.js"></script>
```

Figura 50. Inicialització del controlador del maps.

Per explicar les funcions del controlador del maps, les dividirem en dos blocs: Geo-localització i Esdeveniments.

Geo-localització:

- Funció initMap: Inicialitza el mapa amb unes opcions predeterminades. Centrat en l'usuari (coordenades de latLng) i es crida a la funció que situarà els esdeveniments propers, figura 51.

```
function initMap(){

  var mapOptions = {
    center: latLng,
    zoom: 15,
    mapTypeId: google.maps.MapTypeId.ROADMAP
  };

  map.setCenter(latLng);
  map.setZoom(15);
  map.setMapTypeId(google.maps.MapTypeId.ROADMAP);

  getEvents();

}
```

Figura 51. Inicialització del mapa.

- Funció getEvents: Realitza una petició GET al model "mapa" creant una url amb el nom de l'usuari i les seves coordenades. Ens retorna els esdeveniments més propers, a 1km aprox i filtre els disponibles en aquell moment, figura 52.

```
function getEvents(){

  $http({
    method: 'GET',
    url: ApiEndpoint.url+ 'mapa/usr='+$localstorage.get('name')+'&lat='+latLng.lat+'&lng='+latLng.lng,
  }).then(function successCallback(response) {
      $scope.eventos = [];
      for(var r in response.data) {
        var evento = response.data[r];
        //filtre de dates
        if (evento.ini_date){
          var ahora = new Date()
          fechaE = new Date(evento.exp_date.slice(0,10));
          if (fechaE > ahora){
            $scope.eventos.push(evento);
          }
        }else{
          $scope.eventos.push(evento);
        }
      }
      createMarkers();


  }, function errorCallback(response) {
      console.log("ERROR GetEvents");
  });

}
```

- Funció getCurrentPosition: Aquesta funció la proporciona el plugin de Geo-localització de Cordova i et retorna la posició del dispositiu.

Esdeveniments:

- Funció createMarkers: Utilitza les dades rebudes per getEvents i comprova si els esdeveniments son limitats o parades. Juntament amb “fillInfoWindow” omple el mapa amb una icona depenent el tipus d’esdeveniment.
- Funció fillInfoWindow: S’encarrega de col·locar la informació de cada esdeveniment un cop toquem algun. Se’ns obrirà un petit cuadre de text amb la informació mínima i, en el cas d’un esdeveniment limitat, es podrà accedir als seus detalls.

A grans trets, aquestes son les funcions més importants d’aquest controlador. La resta son funcions secundaries per actualitzar el mapa o controlar el cooldown de les parades, per exemple. Tot el controlador funciona a través de 2 timers. El primer serveix per controlar la posició de l’usuari, amb un refresc curt, 1 segon, mentre que el segon timer s’utilitza per actualitzar la llista d’esdeveniments a mostrar al mapa, que s’actualitza amb menys constància, uns 10 segons.

Compilació

L'última part de la implementació d'aquest projecte es tracta sobre poder aconseguir, amb el mateix codi, una aplicació tant per Android com per IOs.

Al crear el projecte d'Ionic s'afegeixen les plataformes amb les quals voldràs compilar l'app amb les comandes de la figura 53.

```
$ ionic cordova platform add ios
$ ionic cordova platform add android
```

Figura 53. Comandes per afegir les plataformes desitjades.

Per la part d'Android no han aparegut problemes. Introduint la comanda "ionic cordova build android" crea l'apk d'android a una nova carpeta anomenada "platforms/android/build/outputs/apk".

En canvi, pel tema d'IOs ha sigut més complicat ja que la pròpia estructura que proporciona Ionic petava inicialment. Es van canviar llibreries, versions de node i d'xCode i no es podia solucionar.

Consultant altres projectes híbrids, en els fitxers de la carpeta resources, que es pot veure a la figura 6, vaig veure que splash.png tenia un altre mida. Vaig descarregar un altre projecte i substituir el splash.png.

En aquest moment el build es va realitzar, creant una sèrie de fitxers a platforms/ios, figura 54.

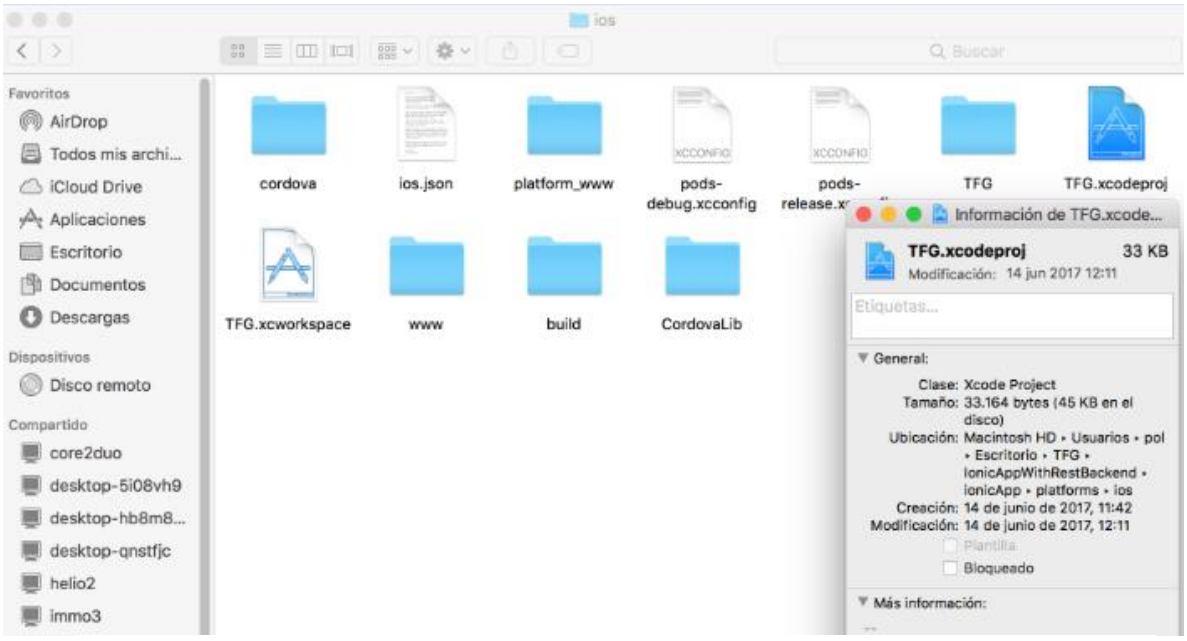


Figura 54. Estructura per l'execució de l'aplicació en IOs

Entre aquests fitxers podem veure un .xcodeproj que serà executat desde el nostre xCode. En l'emulador que proporciona, podem veure que hi ha algun error de maquetació, el qual desquadra algunes icones, figura 55.

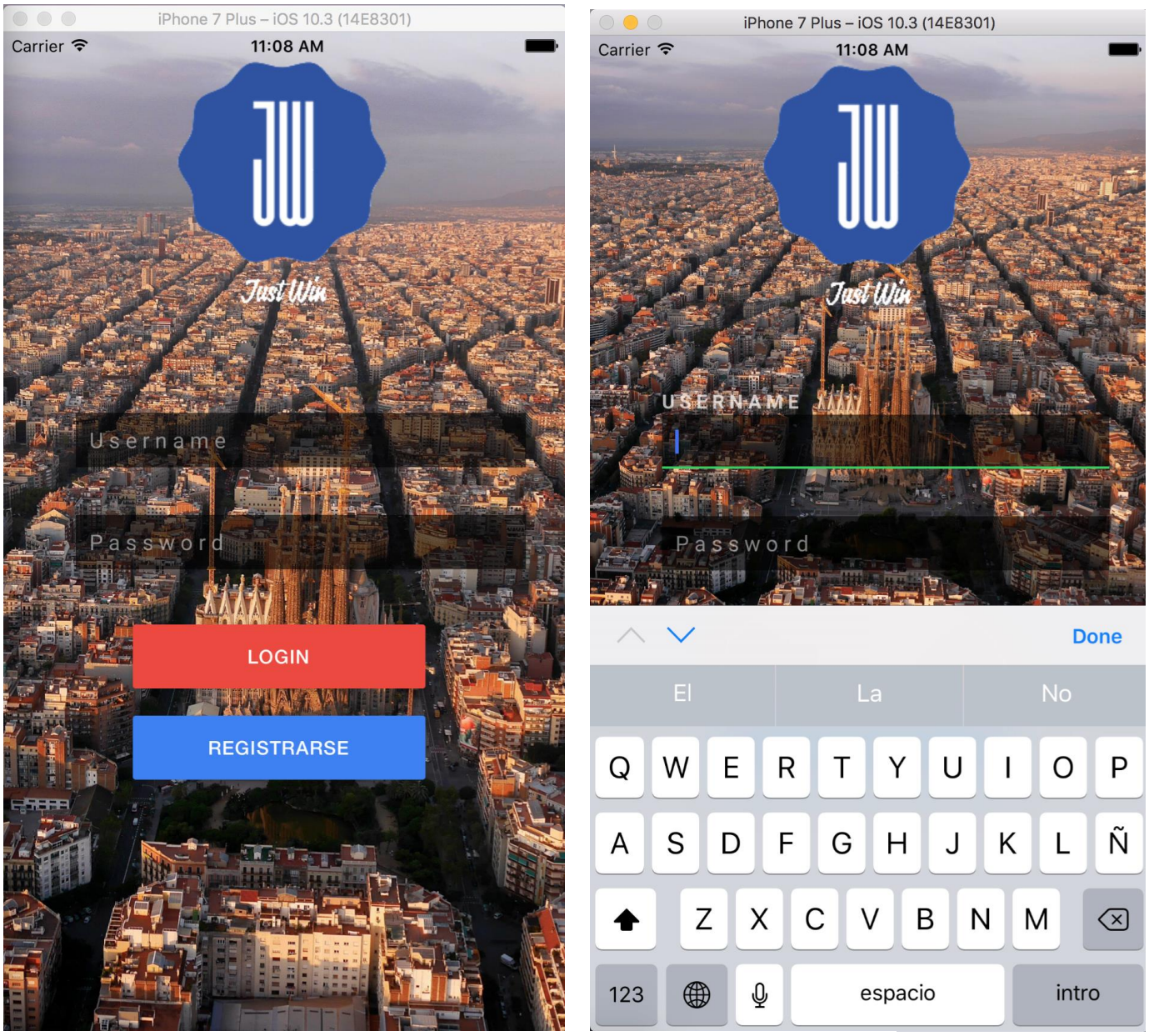


Figura 55. Execució de l'aplicació en un emulador d'Iphone 7 Plus.

6. Continuïtat del projecte

En aquesta versió, l'aplicació JustWin no reporta recompenses reals. Aquest seria un dels objectius a millorar a curt/mig termini .

Actualment es genera un número únic que representa el codi de la recompensa lligat a un nom que indica quina recompensa és. Aquest mètode es podria canviar si tinguéssim codis reals afegint un registre nou amb tots els codis i algun atribut indicant si s'ha entregat o no.

També tenim l'opció de canviar-ho per una recepció d'una imatge amb un codi QR, per exemple, ja que els esdeveniments també estan preparats per poder mostrar imatges.

Per un altre banda, seria imperatiu que funcionés amb ip pública, ja que actualment no pot tenir gaires usos comercials si el seu funcionament es local. Encara que amb algun petit canvi de lògica en l'aplicació, podria ser l'eina per alguns tipus de Room Scape, en els que només es pot utilitzar l'aplicació per resoldre algun enigma o trobar alguna imatge en particular per poder avançar.

Un últim punt a tractar seria el control d'errors. En aquesta versió del projecte, l'usuari només té 2 vistes en les que pot afegir dades: El registre i el perfil.

En els dos cassos, per l'email, es controla a la part client mitjançant un ng-pattern i un missatge, figura 56.

```
<ion-md-input placeholder="Email" highlight-color="balanced" type="email" name="email" ng-pattern="/^[_a-z0-9]+(\.[_a-z0-9]+)*@
[a-z0-9-]+(\.[a-z0-9-]+)*(\.[a-z]{2,4})$/" ng-model="data.email"></ion-md-input>
<span style="color:white; padding-left: 4%;" class="error" ng-show="myForm.email.$error.pattern">Email no válido</span>
```

Figura 56. Control d'email.

Si encara així s'envia un email que no es correcte, el propi Django tira enrere la petició i informem amb un popup de l'error en el registre.

Pel que fa al perfil, s'indicarà en tot moment quina es la dada que està mal introduïda, figura 57.

```
$scope.actualizar = function() {

    var data = {name : $localstorage.get('name'), email : $scope.data.email, password : $scope.data.password,
    var success = false;
    $http({
        method:'POST',
        url: ApiEndpoint.url+'perfilActualizado/',
        data: data,
        headers: {
          'Content-Type': 'application/json; charset=UTF-8',
        }

    }).then(function successCallback(response) {
        success = response.data

        if(success == "true"){
            var alertPopup = $ionicPopup.alert({
              template: 'Se ha modificado su perfil correctamente'
            })
            $state.go('app.home');

        }if(succes == "WRONGPASS"){
            var alertPopup = $ionicPopup.alert({
              title: '<u>Error al modificar datos</u>',
              template: 'Pass incorrectas'
            })
            $state.go('app.home');

        }if(succes == "NOTINFO"){
            var alertPopup = $ionicPopup.alert({
             title: '<u>Error al modificar datos</u>',
              template: 'Hemos tenido un problema, prueba mas tarde. Gracias.'
            })
            $state.go('app.home');
        }

        else{
            var alertPopup = $ionicPopup.alert({
              title: '<u>Error al modificar datos</u>',
              template: 'No se ha podido modificar su perfil.'
            })
        }



    }, function errorCallback(response) {
        console.log("ERROR actualizacion");
    })
```

Figura 57. Control d'email.

Per que la aplicació surtis al mercat, s'hauria de modificar el controls d'errors creant una nova taula a la base de dades i afegint tots els tipus d'errors diferents, per exemple. Des d'alla s'enviarien els missatges d'error cap al client i estaria més centralitzat en el cas d'afegir-ne de nous o modificar els actuals.

7. Conclusions

Ha sigut el treball més complert que he realitzat mai. Per primera vegada m'he introduït de ple a la programació per dispositius mòbils amb una eina la qual hem permet apropar-me a la major part del mercat. És veritat que aconseguir d'aquesta manera un rendiment com el d'una aplicació nativa és complicat, però el temps que t'estalvies en realitzat només un codi, en molts cassos pot ser interessant.

També he de dir, que cara a la programació nativa, per la part d'IOs, i per a mi la més desconeguda més enllà de ser usuari, no he pogut ampliar gaire els coneixements en la seva programació ja que el natiu és completament diferent. En canvi, si que he après força sobre el funcionament dels emuladors d'IOs i sobre xCode, per la banda d'Apple.

En un principi pensava que utilitzar diferents complements dels dispositius mòbils seria complicat però Cordova ho fa molt senzill. Per moltes funcions, t'ofereix un ventall de plugins testejats per la comunitat impressionant, el que també t'obliga ha estar atent a qualsevol actualització, ja que molts plugins s'abandonen i apareixen de nous.

Per "primer" cop, he fet de dissenyador i, encara que no sigui el meu fort, he pogut recrear bona part de les pantalles que m'havia imaginat en un principi, al pensar l'aplicació. També he après força sobre l'ús dels css i en llibreries sobre nous contenidors i sobre animacions.

En aquest projecte no s'utilitzen gaire, però es poden utilitzar un gran número d'animacions diferents fetes per css, encara que també n'hi ha que treballen en js. La compatibilitat de totes les llibreries entre elles m'ha deixat bocabadat, ja que al instal·lar dos diferents, ja pots comprovar que els seus styles son semblants, però aquesta és la màgia de Cordova, et permet unificar moltes llibreries semblants i que no es solapin, ja que ja estan preparades per treballar juntes.