Escola Universitària
Politècnica de Mataró

Centre adscrit a:

UNIVERSITAT POLITÈCNICA
DE CATALUNYA

Enginyeria Tècnica en Informàtica de Gestió

Gestió de clients per Iphone

Memòria

Víctor Sans de Amores
PONENT: Joan Jou Majó

PRIMAVERA 2011

TecnoCampus
Mataró-Maresme

Agraïments

Aquest projecte final de carrera ha estat el de major càrrega que he realitzat en tota la meva vida com a estudiant. En nombroses ocasions he acudit a amics, companys i familiars per demanar ajuda, els quals en tot moment han estat al meu costat, ajudant-me en tot el que he necessitat, o simplement donant-me el suport o ànims que requeria per poder seguir endavant.

A la Cristina Cendrós que en tot moment ha estat al meu costat, per la seva paciència i suport, fent possible que fos capaç de veure que podia aconseguir acabar exitosament el projecte.

Als meus pares Joan i Elena i als meus germans Carla i Jordi per els ànims i suport durant aquests dies tan importants per a mi.

Al meu tutor Joan Jou, que gràcies als seus consells i recomanacions he aconseguit finalitzar aquest projecte.

A tots, moltes gràcies

Resum

“Gestió de clients per Iphone” és una aplicació per als treballadors de l’empresa Proicsa,SL.

Aquest projecte permet portar una gestió automatitzada dels clients, a la vegada que implementa les principals funcions de l’Iphone (Mapes, GPS, fotografies) per facilitar les tasques del treballador.

Es pretén que, aplicant aquest projecte a l’empresa, es millori l’eficiència del treballador, així com l’estalvi de paper i la rapidesa en el servei de cara al client.

Resumen

“Gestió de clients per Iphone” es una aplicación para los trabajadores de la empresa Proicsa,SL.

Este proyecto permite llevar una gestión automatizada de los clientes, a la vez que implementa las principales funciones del Iphone (Mapas, GPS, fotografias) para facilitar las tareas del trabajador.

Se pretende que, aplicando este proyecto en la empresa, se mejore la eficiencia del trabajador, así como el ahorro de papel i la velocidad en el servicio de cara al cliente.

Abstract

"Gestió de clients per iPhone" is an application for workers in the company Proicsa, SL

This project will bring an automatized management of customers, while implementing the main functions of the iPhone (Maps, GPS, pictures) to facilitate the worker's tasks.

It is intended that this project implemented in the enterprise, improve worker efficiency, as well as saving paper and speed of service for the customer.

Índex.

Índex de figures destacades

1.Introducció.

Avui dia, la implantació de les noves tecnologies a l'empresa s'ha tornat quasi indispensable. Les possibilitats que ofereixen, permeten a les empreses aconseguir avantatges competitives vers a les demes companyies.

En el mercat, hi ha infinitat de dispositius i aparells que poden oferir aquestes avantatges, i un d'aquests es l'Iphone. Aquest telèfon mòbil, a part de complir les funcions típiques d'un telèfon (trucades, sms, agenda...), disposa d'un gran ventall de possibilitats que poden aportar noves i eficaces solucions, o millores en apartats que fins ara no es tenien en compte.

Disposar d'aquest dispositiu, permetrà al treballador automatitzar tasques que fins ara es feien manualment, a la vegada que agilitzarà i facilitarà tot el procés de gestió de clients. Es busca millorar el servei que es dóna al client, reduint el temps en els desplaçaments, així com el temps que es triga en omplir totes les dades dels documents que s'han d'entregar als clients.

Per aconseguir aquestes millores, s'ha optat per dissenyar una aplicació per Iphone, que aprofiti les principals característiques que aquest dispositiu brinda a l'usuari, entre les que destaquen el GPS, la connexió a Internet i la càmera fotogràfica.

2. Objectius.

El principal objectiu d'aquesta aplicació és **eliminar del tot el paper** quan es realitza un servei a un client, per tal d'aconseguir un estalvi econòmic. És a dir, quan un treballador acaba la feina que té programada per a un client, aquest hauria d'entregar un albarà, juntament amb un full d'hores emprades i, finalment, una factura. Amb aquesta aplicació el que es pretén és fer-ho tot de forma electrònica, de manera que en cap moment hi intervingui cap full de paper. El fet d'eliminar el paper entre el client i l'empresa, provoca un estalvi de diners que a la llarga és significatiu.

Actualment, la **rapidesa de servei** cap a un client, és un dels aspectes més valorats de les empreses, per això, aquest és el segon objectiu d'aquesta aplicació. Gràcies a ella es podrà millorar l'eficiència amb la que es porta a terme la feina. El client visualitzarà de forma immediata els albarans i les factures pertinents a la feina realitzada. El treballador reduirà el temps emprat en l'ompliment de documents com el de la feina realitzada, ja que serà **quasi automàtica**. A això s'ha de sumar que l'aplicació guiarà al treballador via GPS, el que comportarà una reducció significativa en el temps de desplaçament entre clients. Tota aquesta reducció de temps, es transforma en un estalvi econòmic en forma d'hores treballades, a la vegada que augmentarà el número de clients realitzats per dia

3. Motivació.

Aquesta aplicació es va pensar a partir de passar més de 5 anys treballant a l'empresa Proicsa, S.L. destinada al manteniment i instal·lació de materials contra incendis.

L'activitat principal de l'empresa, el 85% aproximadament, és el manteniment de les instal·lacions fetes a diferents clients. Aquesta feina es basa en efectuar les pertinents revisions d'extintors i instal·lacions de detecció i extinció instal·lades. Aquesta labor la realitza un sol treballador, el qual és l'encarregat de revisar la instal·lació i posteriorment fer entrega del parte de treball.

Es va pensar que si s'apliquessin les noves tecnologies al dia a dia d'aquest treballador, es reduiria el temps que triga en realitzar cada client, a la vegada que l'empresa estalviaria diners i temps per cada client realitzat. Per poder aconseguir aquesta millora, s'ha pensat en crear una aplicació per Iphone, la qual utilitzarà el treballador en el seu entorn de treball.

Abans de dissenyar l'aplicació s'ha de separar què volem aconseguir. S'ha desglossat en dues parts ben diferenciades entre sí:

- Disminució del temps destinat a cada client (des de que comença la feina, fins que aquesta es cobra) el qual comporta un estalvi econòmic en forma d'hores emprades pel treballador, i un augment dels clients realitzats per dia.
- Estalvi de diners en minimitzar la utilització de paper per cada client realitzat.

Un cop conegut que volem aconseguir, s'han de buscar quines accions provoquen que això no es compleixi.

S'analitza el que fa actualment un treballador a partir de que se li assignen uns clients a realitzar:

- El treballador rep de l'empresa un full amb la ruta a seguir (diferents clients a realitzar durant un dia) que serveix per tenir a primera vista una llista de totes les direccions a les que ha d'anar, així no ha de consultar la fitxa individual de cada client. A part d'aquest full, també rep la fitxa individual del client, en la que hi

consten totes les dades del client, una relació de tots els extintors dels que disposa, i si te sistema de detecció i extinció d'incendis.

- El següent pas a seguir, és anar cap al client. El treballador decideix quin client realitzarà primer, normalment intentant deduir quin esta més a prop fent servir una guia de carrers, això comporta que el treballador triga força temps en decidir a quin client va, i que més d'una vegada, el treballador es perdi o tingui dificultats per arribar al seu destí.

Si s'analitza què passa quan un treballador finalitza la feina d'un client determinat, trobem que:

- El treballador només deixa com a comprovant al client, un full en el que hi constava la feina realitzada amb el total d'hores emprades, i un esborrany de l'albarà. Depenent del volum de feina, un treballador trigava uns 20 minuts per client en emplenar aquestes dades. El temps que passa des de que el treballador finalitza la feina fins a que l'empresa envia l'albarà en net i la factura podia ser de 7 a 10 dies, temps que triga correus en enviar una carta, això provoca que el cobrament de la feina s'alenteixi considerablement.

- El número de fulls emprats al final per a cada client és d'un mínim de 9 fulls i un màxim determinat pel volum de feina realitzada. Els 9 fulls es desglossen en tres còpies per cadascun dels documents: full d'hores, albarà i factura. De les 3 copies una era per al client, la segona per a l'arxiu privat de l'empresa i la tercera per a la còpia que s'ha d'entregar a hisenda.

Un cop analitzats els diferents punts que estan subjectes a millora, s'ha de plantejar que es fa i com es fa.

- Es crearà una aplicació per a Iphone, que permetrà emplenar els partes de treballs, i mostrar al client els albarans i factures pertinents. L'aplicació a la vegada disposarà d'un navegador GPS que crearà les rutes entre els clients que es volen realitzar, i la possibilitat de realitzar fotografies per deixar constància de possibles incidències amb les que el treballador es pugui trobar.

- Si s'aconsegueix mostrar al client tota la documentació abans esmentada al moment de finalitzar la feina, i s'hi suma que l'emplenament de dades es quasi automàtic i que no es fa servir cap document físic (fulls de paper) de cara al client (només s'utilitza una copia de cada document per a la comptabilitat), podem dir que s'assoliran els objectius del projecte, que és l'estalvi de paper, temps i diners.

Un cop s'han analitzat els diferents punts que estan subjectes a millora, es pot dir que s'està preparat per a començar a dissenyar l'aplicació. Aquesta etapa bàsicament s'ha de dedicar a escriure tot el codi de programació necessari perquè l'aplicació funcioni correctament.

4. Estudi de mercat.

Actualment no existeix cap aplicació per a Iphone que realitzi aquestes funcions, la qual cosa permet afirmar que no hi ha cap altra opció a la que comparar-se. Existeixen aparells destinats a la signatura de rebuts, com els que fan servir els repartidors de paqueteria urgent, però només serveixen per donar la conformitat de la rebuda del mateix, no permeten realitzar les tasques abans mencionades.

Aquesta aplicació tracta de satisfer unes necessitats, està feta a mida per a un cas concret, per tant, poques possibilitats hi ha de que alguna aplicació existent pugui aportar les solucions desitjades.

El cost del aparell pot suposar un inconvenient al principi, però si es pensa en l'estalvi que proporcionarà a la llarga, i el fet de poder realitzar la feina d'una manera molt més eficient i ràpida, aquesta aplicació surt rentable.

5. Metodologia i tecnologia.

5.1. Tecnologia i entorn de desenvolupament.

El projecte està desenvolupat amb els llenguatge de programació Objective-C i PHP. El primer és per a desenvolupar l'aplicació d'Iphone, el segon per l'elaboració dels continguts web.

A continuació s'especifica ,detalladament, tot el software utilitzat per a la realització del projecte:

- Sistema operatiu Mac OSX.
- Processador de text Microsoft Word for Mac 2011.
- Servidor web i de base de dades: MAMP (Mac, Apache, MySql, PHP).
- Entorn de desenvolupament d'aplicacions Xcode 3.2.5.
- Entorn de desenvolupament web Dreamweaver.
- Simulador d'Iphone IOS 4.2.

Abans de començar a programar s'han identificat els requeriments funcionals, per a fer-ho s'han usat els Casos d'ús.

El software per desenvolupar l'aplicació Iphone ha estat Xcode, i per l'aplicació web s'ha usat Dreamweaver.

A l'hora de programar l'aplicació s'ha usat Objective-C que és el que es requereix per a programar dispositius Iphone. Aquest usa la programació orientada a objectes, i el patró Model Vista Controlador (MVC). Aquest patró per definició aïlla el model de dades, la interfície usuari i la lògica de control. Per als apartats web s'ha usat PHP i MySQL.

5.2. Requisits tecnològics.

Es necessari disposar d'un servidor Apache capaç de resoldre pàgines dinàmiques PHP. Aquest servidor també ha de ser capaç d'allotjar bases de dades MySQL. PHP i MySQL s'adapten a la perfecció, així que no hi haurà cap problema a l'hora de fer consultes des de PHP a la base de dades.

Per poder tenir tot això instal·lat s'ha optat per fer servir un MAMP (Mac, Apache, MySQL, PHP) , és una plataforma open source per a Mac, la qual s'instala automàticament i és molt senzilla de configurar. Tanmateix està disponible per a Windows (WAMP) i Linux (LAMP).

També és necessari disposar d'un dispositiu Iphone, el qual es capaç d'interpretar l'aplicació i totes les funcionalitats que aquesta fa servir (GPS, Càmera...).

6. Llenguatges de programació.

Com ja s'ha comentat, aquest projecte usa, bàsicament, 2 llenguatges de programació:

- Objective C
- PHP

A continuació farem una breu descripció de cadascun d'ells.

6.1. Objective C.

Objective C és una extensió de C per convertir-lo en un codi orientat a objectes. És més net, petit i fàcil d'aprendre que C. Aquest llenguatge és l'utilitzat per Mac OS X.

El llenguatge no està estandaritzat per cap organisme internacional, Apple y NEXTSTEP són els que han contribuït a crear aquest llenguatge. Està basat en Smalltalk, un dels primers llenguatges de programació orientats a objectes.

És un llenguatge molt dinàmic, moltes decisions es prenen en temps d'execució. Els atributs d'una classe no tenen perquè estar tipificats.

El marc de treball de desenvolupament d'aplicacions Objective-C per a Mac OS X és Cocoa. Cocoa és una biblioteca extensa. Inclou molts marcs de treball (frameworks) de programari que contenen definicions dels objectes que pots utilitzar o adaptar a les necessitats del teu programa. Aquesta inclou un marc de treball Elemental, un marc de treball de Kit d'Aplicacions (per construir la interfície gràfica d'usuari), entre d'altres.

Mac OS X també inclou eines de desenvolupament per ensamblar aplicacions. Aquests són l'Interface Builder, un programa que permet el disseny d'una aplicació gràficament i l'ensamblat interactiu amb l'interfície d'usuari, i l'Xcode, un programa de gestió de projectes que proporciona accés gràfic sobre el compilador, el depurador, la documentació, un editor de programes, i altres eines.

El llenguatge Objective-C ha escollit el marc de treball Cocoa per varies raons. La primera i més important, que és un llenguatge orientat a objectes. El tipus de funcionalitat que està empaquetada en el marc de treball Cocoa només pot obtindre's mitjançant tècniques orientades a objectes.

Segon, perquè l'Objective-C és una extensió de l'estandard ANSI C, els programes C existents poden adaptar-se per utilitzar-los en els marcs de treball sense perdre gens de feina feta en els desenvolupaments originals. Com que Objective-C incorpora C, obtens tots els beneficis de C quan treballes amb l'Objective-C. Pots escollir quan fer quelcom mitjançant l'orientació a objectes (definint una nova classe, per exemple) i quan utilitzes tècniques de programació procedimental (defineixes una estructura i algunes funcions incloses d'una classe).

A continuació un esquema de l'arquitectura de Cocoa:

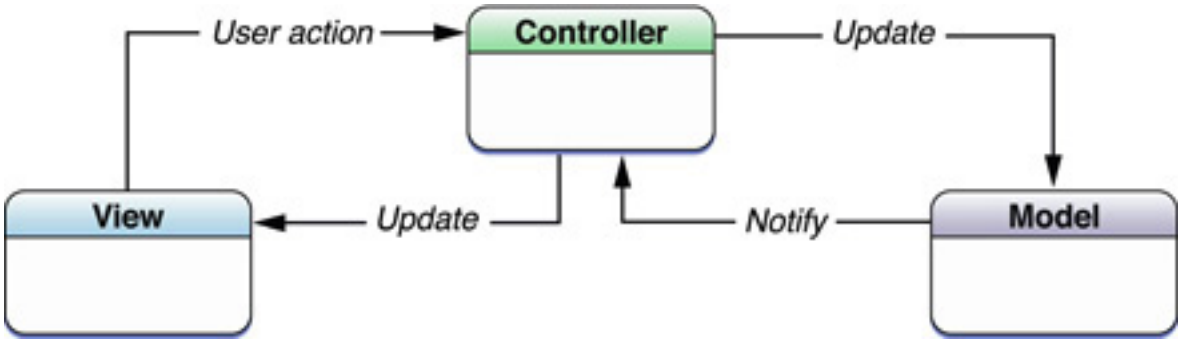


Fig. 6.1. Arquitectura Cocoa.

Com es pot observar en la figura anterior, Cocoa usa el patró Model-Vista-Controlador (MVC). En l'àmbit de Cocoa, l'MVC és un patró molt important. Les principals millores que Apple ha fet amb Cocoa, és a dir, enllaços i dades bàsiques, són manifestament basats en el patró MVC.

6.2. PHP.

El PHP és un acrònim que significa (***P**HP **H**ypertext **P**re-Processor*). Inicialment el nom va ser PHP Tools. Es tracta d'un llenguatge interpretat d'alt nivell inclòs en pàgines web HTML i executat en un servidor.

El PHP té la capacitat de ser executat en la majoria de sistemes operatius, així com Mac OSX, Linux, Windows..., i pot interactuar amb els servidors web més populars, ja que existeix en versió CGI, mòdul per Apache i ISAPI.

Té la capacitat de connexió amb la majoria dels motors de bases de dades que s'utilitzen en l'actualitat, destacant la connectivitat amb MySQL que és la que fem servir en aquest projecte.

El PHP no obliga al programador a seguir una determinada metodologia a l'hora de programar. El programador pot aplicar qualsevol tècnica de programació i desenvolupament que li permeti escriure un codi lògic, ordenat i estructurat.

A continuació un esquema del funcionament del PHP:

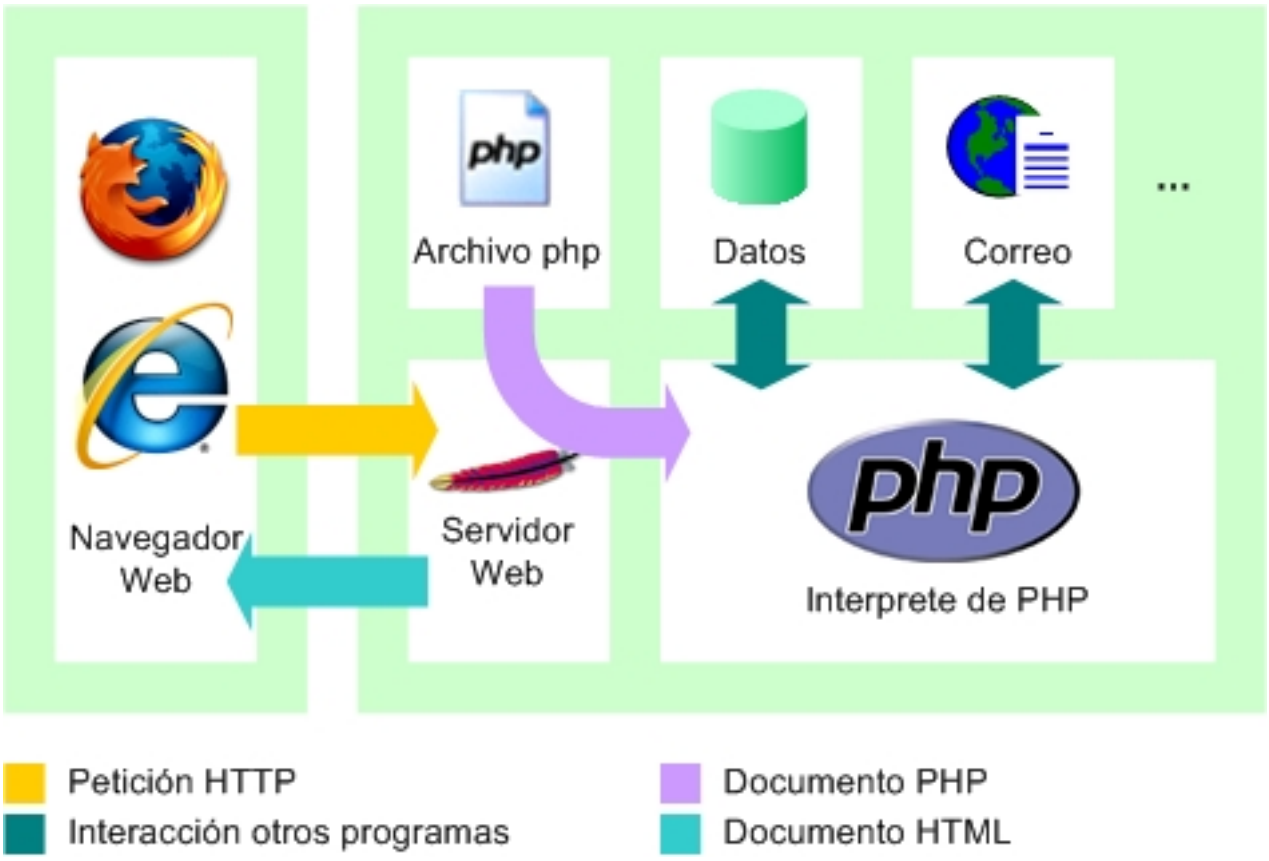


Fig. 6.2. Esquema de funcionament PHP extreta de www.kamlov.site90.net.

7. Eines de programació.

7.1. Macromedia Dreamweaver.

El Macromedia Dreamweaver és un editor dinàmic HTML de pàgines web. A part de permetre l'escriptura del codi de programació té l'opció de visualitzar, al moment, tot allò que s'està programant. Per tant, es pot dir que aquesta és una eina útil tant per a dissenyadors com per a programadors. Amb aquesta eina és possible treballar simultàniament amb codi HTML per a crear la base de la web, Javascript per a crear els elements necessaris, CSS per a donar un estil gràfic i visual òptim, i, el més important, permet desenvolupar aplicacions que s'estiguin executant en un servidor i d'aquesta manera poder accedir a les dades i continguts de forma local, en aquest cas es farà amb PHP i MySQL.

Dreamweaver permet l'escriptura de codi de programació d'una forma clara i neta. L'editor de codi és capaç d'interpretar la presència de diferents codis de programació web en una mateixa plantilla, i deferencia mitjançant colors i estils el codi escrit, això facilita molt la comprensió i escriptura del codi escrit. Cal recordar, que Dreamweaver és l'eina d'edició web més utilitzada per tot tipus d'usuaris.

Després de veure què aporta aquesta aplicació, es pot afirmar amb tota seguretat, que és l'eina que millor servei donarà per a desenvolupar la part en la que es fa servir aquesta tecnologia.

7.2. Xcode.

Xcode és l'entorn de desenvolupament integrat (IDE) de Apple Inc. el qual es subministra gratuïtament junt amb Mac OSX. Xcode treballa conjuntament amb Interface Builder, una eina gràfica per a la creació d'interfícies d'usuari.

Ofereix una interfície atractiva i potent per a crear i administrar projectes de desenvolupament de software per Mac OSX (en el nostre cas Iphone).

Xcode també incorpora un simulador d'IOS d'Iphone, el qual permet simular Iphone o Ipad en el nostre Mac, facilitant la visualització del nostre projecte a mida que anem avançant.

Aquest IDE és, avui en dia, l'únic que permet el desenvolupament d'aplicacions per a Mac OSX. En aquest cas s'ha desenvolupat una aplicació per a un dispositiu Iphone, per tant, no queda cap altre remei que fer servir aquest entorn.

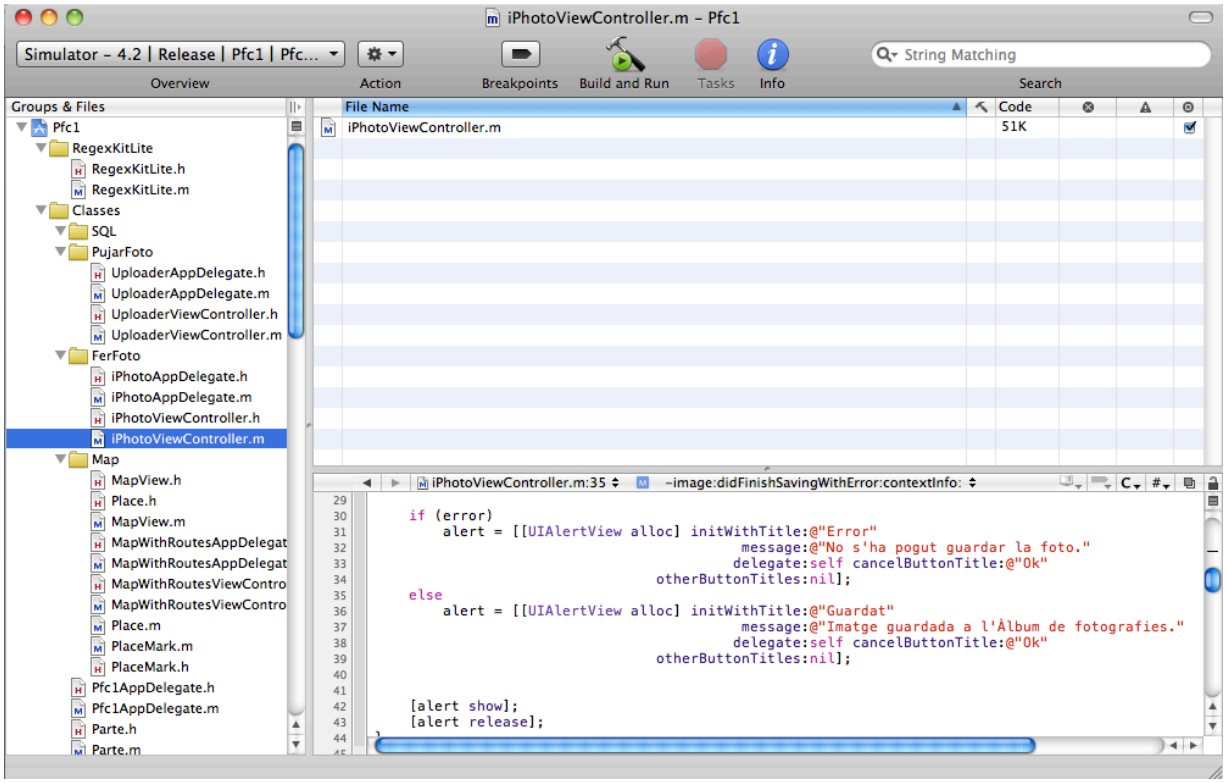


Fig. 7.2. Entorn Xcode.

7.3. Altres eines.

7.3.1. MySQL Workbench.

MySQL Workbench és un software creat per l'empresa informàtica Sun Microsystems, que permet modelar diagrames entitat-relació per a bases de dades MySQL. Pot utilitzar-se per a dissenyar l'esquema d'una base de dades nova, documentar una existent o realitzar una migració complexa.

L'aplicació elabora una representació visual de taules, vistes, procediments emmagatzemats i claus foranies de la base de dades. A més, es capaç de sincronitzar el model de desenvolupament amb la base de dades real, enginyeria inversa per a importar l'esquema d'una base de dades ja existent, o "forward engineer" el qual a partir d'un model entitat relació, crea el codi SQL necessari per a crear la base de dades.

En aquest cas, s'ha creat el diagrama de la base de dades, amb totes les taules i les seves relacions. Un cop creades s'ha usat "forward engineer" connectat a la vegada amb el servidor MySQL. El resultat que s'obté és la base de dades perfectament creada.

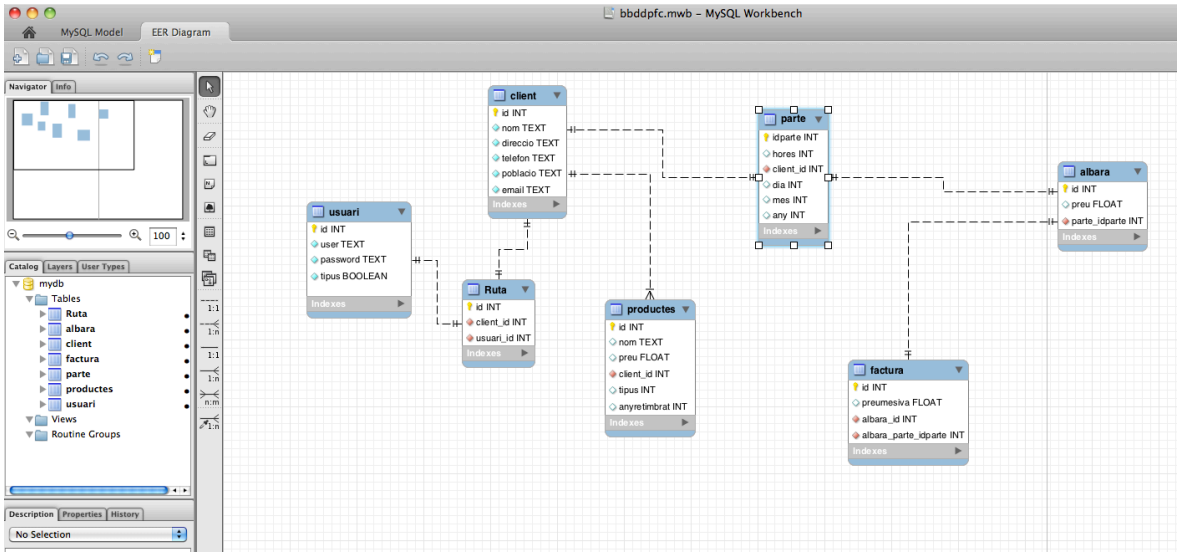


Fig. 7.3. MySQL Workbench.

7.3.2. PhpMyAdmin.

PhpMyAdmin és un programa que permet accedir a totes les funcions d'una BBDD MySQL mitjançant una interfície web.

PhpMyAdmin és un programa de lliure distribució que es pot fer servir des de qualsevol ordinador. Per accedir-hi només cal escriure en un navegador web el domini seguit de /phpmyadmin. En aquest cas l'accés seria: http://localhost:8888/phpMyAdmin/

L'aplicació permet crear taules, inserir dades en taules existents, navegar pels diferents registres, editar-los, esborrar-los... També permet executar sentències SQL per a realitzar totes les accions que es necessitin. Amb aquesta aplicació és possible també rebre sentències SQL des d'altres softwares per tal de que es generi automàticament la base de dades pertinent. En aquest cas PhpMyAdmin rebrà la sentència generada per MySQL Wokbench.

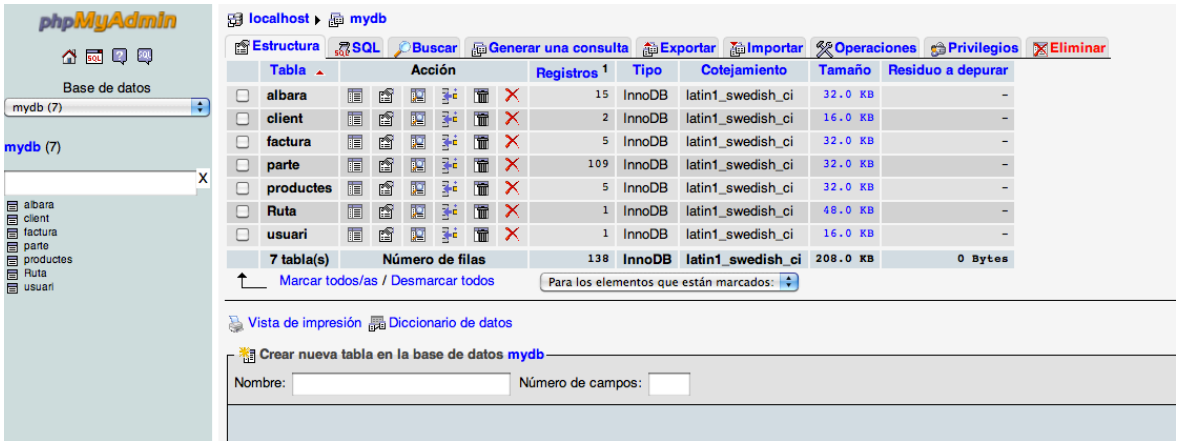


Fig. 7.4. Phpmyadmin.

7.3.3. MAMP.

L'acrònim MAMP fa referència al conjunt de programes software usats per a desenvolupar aplicacions web dinàmiques sobre sistemes operatius Apple MAC OS X.

- **M**ac OS X: Sistema operatiu.

- **A**pache: Servidor Web.

- **M**ySQL: Sistema Gestor de Bases de Dades.

- **P**HP: Llenguatges de programació usats per a la creació d'aplicacions web.

L'avantatge d'usar MAMP, és que incorpora una interfície gràfica que facilita molt la instalació i configuració de tots els serveis necessaris. Un cop instal·lada, només cal que obrim l'aplicació per a que tots els serveis s'activin automàticament. A més, s'ha d'afegir que és una eina de lliure distribució.

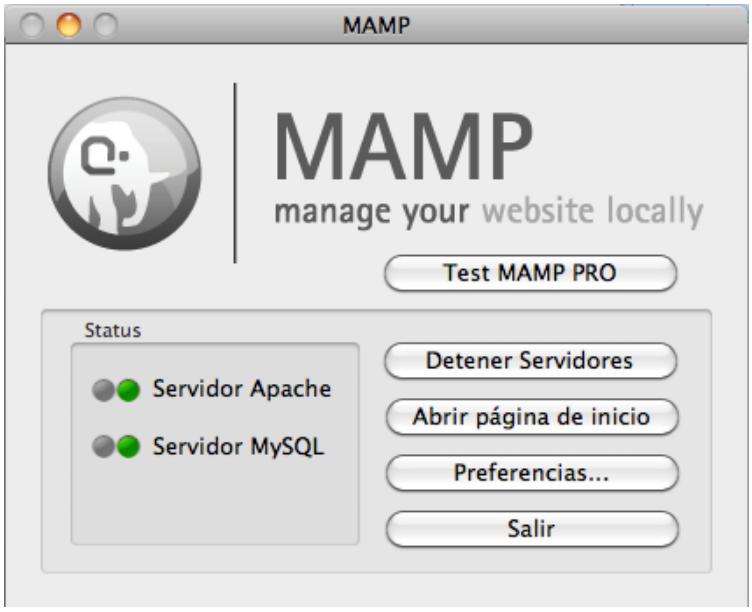
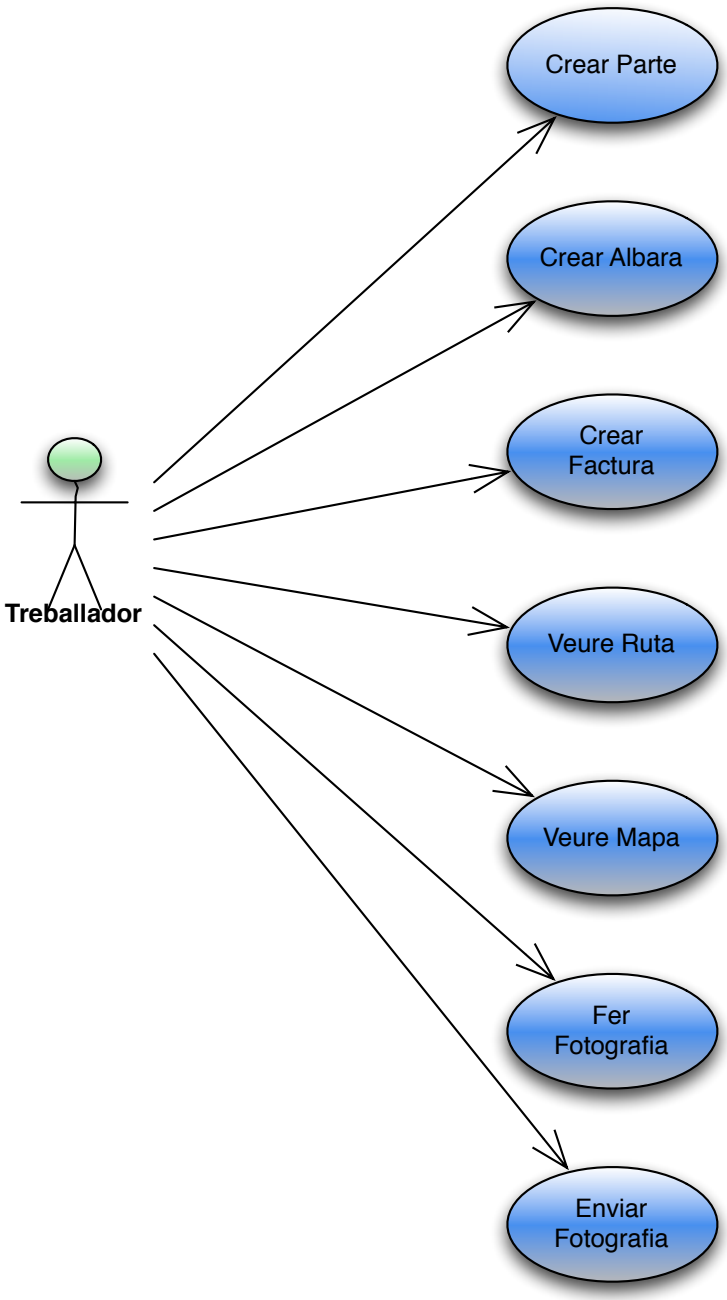


Fig. 7.5. MAMP.

8. Casos d'ús.

8.1. Diagrama de casos d'ús.



8.1.1. Cas d'ús : Crear parte.

Actor involucrat: Treballador

Pre-Condició

El treballador ha d'estar en el menú "Client" de l'aplicació.

Flux normal

1- L'usuari escull l'opció "crear parte" del ventall d'opcions de les que disposa en el menú principal.

2- L'aplicació presenta un ventall d'opcions que el client haurà de seleccionar.

3- El treballador escull a quin client pertany aquest parte.

4- El treballador escull el número d'hores dedicades a aquest client.

5- El treballador introdueix la data en la que s'ha fet efectiva la feina.

6- El treballador confirma les dades fent click al boton enviar.

7- El parte és creat i guardat correctament a la base de dades.

8- El treballador és enviat a la pantalla d'albarà.

8.1.2. Cas d'ús : Crear albarà.

Actor involucrat: Treballador

Pre-Condició

El treballador ha d'haver creat un parte de treball correctament.

Flux normal

1- L'aplicació mostra l'albara al treballador.

2- El treballador comprova que les dades de l'albarà són correctes.

3- El client dóna el vist-i-plau a l'albarà introduint-hi la contrasenya.

4- El treballador fa click al botó "Acceptar albarà" per a confirmar les dades.

5- L'albarà és creat i guardat correctament a la base de dades.

6- El treballador és enviat a la pantalla de la factura.

8.1.3. Cas d'ús : Crear factura.

Actor involucrat: Treballador

Pre-Condició

El treballador ha d'haver creat un albarà correctament.

Flux normal

1- L'aplicació mostra la factura al treballador.

2- El treballador comprova que les dades de la factura són correctes.

3- El treballador fa click al botó "Acceptar factura" per a confirmar les dades.

4- La factura és creada i guardada correctament a la base de dades.

5- El treballador és informat de que la factura s'ha guardat correctament.

8.1.4. Cas d'ús : Veure ruta.

Actor involucrat: Treballador

Pre-Condició

El treballador ha d'estar en el menu "Treballador" de l'aplicació.

Flux normal

1- L'aplicació mostra el menú amb les diferents opcions.

2- El treballador fa click al botó "Ruta".

3- L'aplicació mostra per pantalla els clients assignats al treballador que han de ser realitzats.

4- El treballador pot fer click a un client per veure les dades personals d'aquest.

5- El treballador clica el botó "Veure Ruta" per veure la ruta creada a partir de totes les adreces dels clients assignats.

Flux alternatiu

4.1- El treballador pot tornar en qualsevol moment a la pantalla anterior per veure les dades d'un altre client fent click al botó Ruta.

4.2- El treballador pot tornar al menú principal de l'aplicació fent click al botó menú principal.

8.1.5. Cas d'ús : Veure mapa.

Actor involucrat: Treballador

Pre-Condició

El treballador ha d'estar en el menu "Treballador" de l'aplicació.

Flux normal

1- L'aplicació mostra el menú amb les diferents opcions.

2- El treballador fa click al botó "Mapa".

3- L'aplicació mostra per pantalla diverses caixes de text per a que el client introdueixi direccions a les que vol anar.

4- El treballador fa click al boto "Ruta" per coneixer les adreces dels clients en el cas de que en necessiti alguna.

5- El treballador apreta el botó "Menu Principal" per ser retornat a la pantalla d'introducció de direccions.

6- El treballador introdueix la primera direcció a la primera caixa de text.

Flux alternatiu

4.1- El treballador haurà de repetir aquest pas tantes vegades com clients hagi de realitzar.

8.1.6. Cas d'ús : Fer fotografia.

Actor involucrat: Treballador

Pre-Condició

El treballador ha d'estar en el menu "Treballador" de l'aplicació.

Flux normal

1- L'aplicació mostra el menú amb les diferents opcions.

2- El treballador fa click al botó "Foto incidència".

3- El treballador fa click al botó "Fer fotografia" per accedir a la càmera.

4- El treballador ha de clicar el botó amb dibuix de càmera per a realitzar la fotografia de la incidència.

5- Un cop feta la fotografia es mostra per pantalla 2 opcions.

6- El treballador torna al menú principal fent click al botó "Menú Treballador".

Flux alternatiu

5.1- El treballador fa click al botó "retake" per tornar a fer la fotografia.

5.2- El treballador fa click al botó "use" per acceptar la fotografia feta.

5.2.1- Es mostra per pantalla un missatge confirmant que la fotografia ha estat guardada correctament a l'àlbum de fotografies.

8.1.7. Cas d'ús : Enviar fotografia.

Actor involucrat: Treballador

Pre-Condició

El treballador ha d'estar en el menu "Treballador" de l'aplicació.

Flux normal

1- L'aplicació mostra el menú amb les diferents opcions.
2- El treballador fa click al botó "Enviar foto".
3- El treballador fa click al botó "Seleccionar" per accedir a la galeria de fotografies.
4- El treballador selecciona la fotografia que desitja enviar al servidor fent click sobre ella.
5- L'aplicació mostra la fotografia seleccionada.
6- El treballador fa click al botó "enviar" per enviar la fotografia al servidor.
7- Un cop enviada el boto "enviar" tornarà a estar de color blau.
8- El treballador torna al menú principal fent click al botó "Menú Treballador".

9. Base de dades.

Per fer la base de dades s'ha utilitzat MySQL. S'ha utilitzat MySQL ja que és la que millor s'adapta al PHP, i es fa servir exclusivament per a interactuar amb ella des de les pàgines PHP. En la següent imatge es pot veure el model de la base de dades creada.

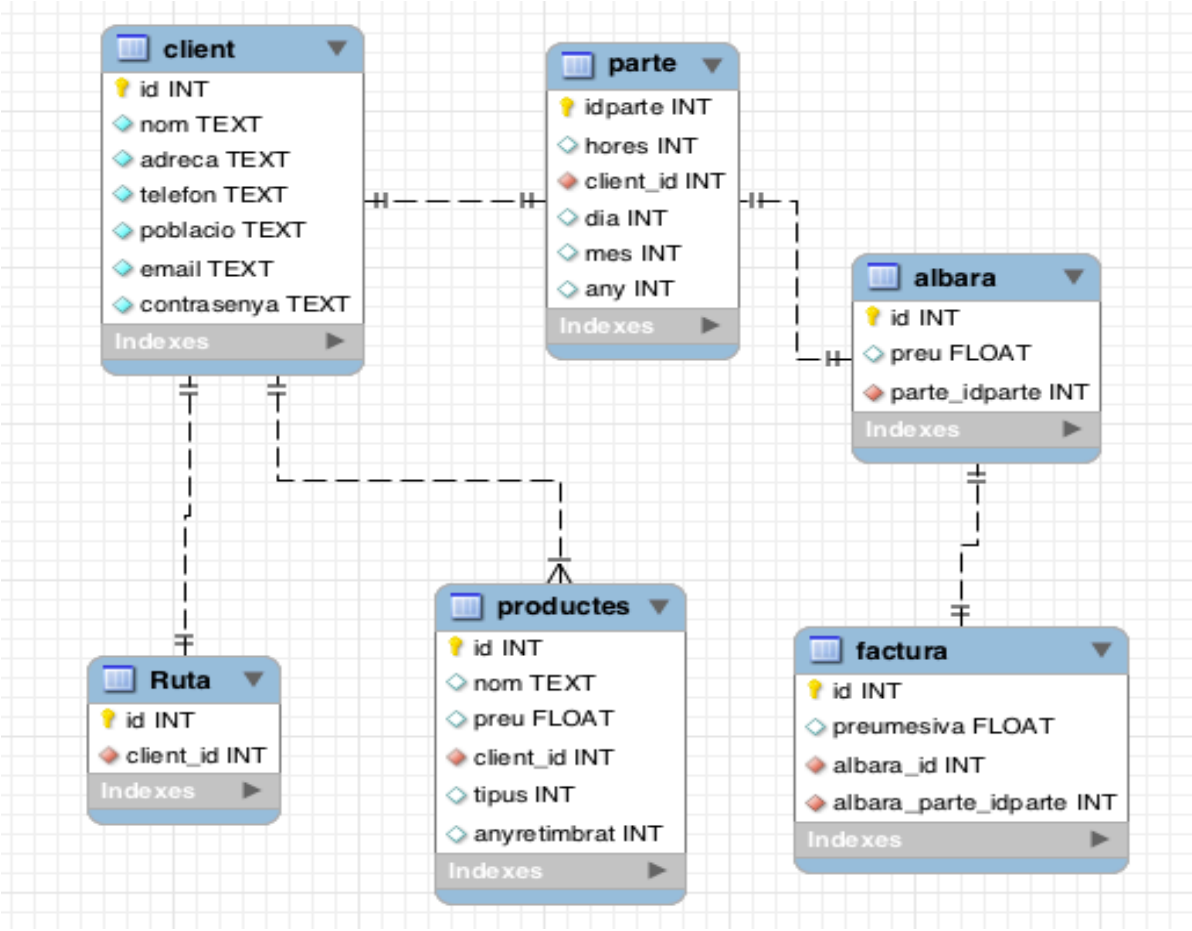


Fig. 9.1. Model de la base de dades.

En ell s'hi troben les següents taules:

- Client.
- Ruta.
- Productes.
- Parte.
- Albarà.
- Factura.

Taula client:

En ella s'hi troben totes les dades referents al client, les quals s'usaran per conèixer l'adreça al moment de crear la ruta dins el mapa, i també per a crear els corresponents partes, albarans i factures quan sigui necessari.

Taula ruta:

Aquesta taula s'utilitza per a guardar la ruta que ha de seguir un treballador en un dia concret.

Taula productes:

En ella s'hi troben les dades referents als productes. Els productes fan referència a un client, ja que cada producte s'ha venut amb anterioritat, per tant, cada producte és d'un client concret.

Taula parte, albarà i factura:

Les tres taules compleixen la mateixa missió, guardar les dades de partes, albarans i factures respectivament. Cadascuna fa referència a l'anterior, d'aquesta manera sempre tenim disponibles les dades de la primera de totes, el parte.

10. Gestió de clients per Iphone.

10.1. Accés a l'aplicació.

Per accedir a l'aplicació del projecte, l'usuari ha de disposar d'un dispositiu Iphone i connexió a Internet (3G o WiFi).

El treballador haurà de navegar pel menú d'aplicacions de l'Iphone fins a trobar l'aplicació corresponent. Un cop trobada només ha de fer un click amb el dit per obrir-la i executar-la.

Fig. 10.1. Accés a l'aplicació.

10.2. Panell principal – Recursos del treballador.

Un cop l'usuari ha accedit correctament a l'aplicació, s'accedeix al menú principal de l'aplicació. S'ha establert com a vista principal la dels recursos del treballador, ja que quan el treballador accedeixi a l'aplicació, les primeres accions que realitzarà estan situades en aquesta vista.

En aquest menú es poden identificar tres parts diferents:

- Menú superior (Navigation Controller).
- Menú inferior (Tab Bar Controller).
- Vista principal (View).

Fig. 10.2. Menú Principal.

Menú superior

El menú superior, a part de mostrar el títol, té una utilitat ben clara, aquesta és la de permetre la navegació a mida que anem avançant per l'aplicació. Per crear aquest menú superior, s'ha utilitzat la classe UINavigationController. La classe UINavigationController implementa un controlador de vista especialitzat que gestiona la navegació de contingut jeràrquic. Aquesta interfície de navegació permet presentar les dades de manera eficient i també fa més fàcil per a l'usuari navegar pel contingut.

```
UINavigationController *navigationController;
```

Fig.10.3. Declaració de la utilització de UINavigationController.

```
[self.window addSubview:navigationController.view];
```

Fig. 10.4. Afegim a la vista un UINavigationController.

A continuació, com mostra la figura 10.5, es pot veure la navegació entre pantalles mitjançant el UINavigationController.

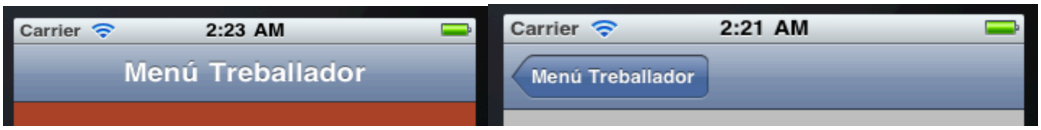


Fig. 10.5. UINavigationController Navegació entre pantalles.

Menú inferior

El menú inferior, té una utilitat ben clara, aquesta és la de presentar una interfície que permet a l'usuari triar entre diferents modes d'operació. Per crear aquest menú inferior, s'ha utilitzat la classe UITabBarController. La classe UITabBarController implementa un controlador de vista especialitzat que gestiona una interfície de selecció de ràdio d'estil.

Aquesta interfície mostra una barra de pestanyes a la part inferior de la finestra per seleccionar entre els diferents modes i per mostrar els diferents punts de vista d'aquesta aplicació.

```
UITabBarController *tabBarController;
```

Fig. 10.6. Declaració UITabBarController.

```
[self.window addSubview:tabBarController.view];
```

Fig. 10.7. Afegim a la vista un UITabBarController.

En aquesta aplicació (com es pot observar en la figura 10.5) s'ha usat per separar en dues parts les possibilitats del treballador, així s'aconsegueix dividir per una part la gestió de clients (Pestanya client), i per altra banda els recursos de que disposa el treballador per a portar a terme la seva feina (Pestanya treballador).

Fig. 10.8. UITabBarController Menú inferior.

Vista Principal

La vista principal implementa la classe UIView, aquesta defineix una àrea rectangular a la pantalla i les interfícies de la gestió del contingut en aquesta zona. En temps d'execució, un objecte de vista s'encarrega de la presentació de qualsevol contingut en la seva àrea i també s'ocupa de les interaccions amb aquest contingut. El marc UIView també inclou un conjunt de subclasses estàndard que pot utilitzar, que van des de simples botons a taules complexes.

En la vista principal del menú principal hi podem observar 4 botons (Figura 8.6). Cada botó ens enviarà a les diferents funcions de les que disposa aquesta aplicació.

Per ordre d'aparició són:

- Ruta.
- Mapa.
- Foto incidència.
- Enviar foto.

Fig. 10.9. Vista principal.

10.2.1. Ruta.

Aquest apartat és un dels més importants de l'aplicació. La ruta, és el nom que es dóna a la feina que ha de realitzar un treballador. La primera acció que realitzarà un treballador a l'inici de la seva jornada laboral, és veure quins clients li han estat assignats.

Al clicar en el botó "Ruta" del menú principal, l'aplicació enviarà al treballador a una nova vista en la qual hi visualitzarà els diferents clients que li han estat assignats.

Aquesta vista fa ús de la classe UITableViewController. La classe UITableViewController crea un objecte de controlador que gestiona una vista de taula. S'ha fet servir aquest tipus de classe, ja que és la més adequada a l'hora de mostrar una llista d'elements, en aquest cas clients.

```
@interface RootViewController : UITableViewController {
```

Fig. 10.10. Declaració classe UITableViewController.

Per omplir aquesta taula, és necessari accedir a la base de dades. El primer que s'ha de fer, és declarar l'arxiu de la base de dades, per a que l'aplicació tingui un "path" on buscar.Un cop s'ha delcarat, s'haurà de seleccionar les dades que interessa mostrar, per últim, només caldrà introduir-les dins del UITableViewController.

```
- (void) copyDatabaseIfNeeded {

NSFileManager *fileManager = [NSFileManager defaultManager];
NSError *error;
NSString *rutaBD = [self getDBPath];
BOOL correcte = [fileManager fileExistsAtPath: rutaBD];

if(!correcte) {

NSString *predRutaBD = [[[NSBundle mainBundle] resourcePath]
stringByAppendingPathComponent:@"client.sqlite"];
correcte = [fileManager copyItemAtPath: predRutaBD toPath: rutaBD error:&error];

    }
}
```

Fig. 10.11. Declaració de l'arxiu de la base de dades.

```
+ (void) getInitialDataToDisplay:(NSString *) rutaBD {

        SQLAppDelegate *appDelegate = (SQLAppDelegate *)[[UIApplication
sharedApplication] delegate];

        if (sqlite3_open([rutaBD UTF8String], &database) == SQLITE_OK) {

          const char *sql = "select id, nom, direccio from client";
          sqlite3_stmt *selectstmt;
          if(sqlite3_prepare_v2(database, sql, -1, &selectstmt, NULL) == SQLITE_OK) {

                    while(sqlite3_step(selectstmt) == SQLITE_ROW) {

                              NSInteger primaryKey = sqlite3_column_int(selectstmt, 0);
                              Client *clientObj = [[Client alloc] initWithPrimaryKey:primaryKey];
                              clientObj.clientName = [NSString stringWithUTF8String:(char
*)sqlite3_column_text(selectstmt, 1)];
                              clientObj.adreca = [NSString stringWithUTF8String:(char
*)sqlite3_column_text(selectstmt, 2)];
                              clientObj.isDirty = NO;

                              [appDelegate.clientArray addObject: clientObj];
                              [clientObj release];
                    }
          }
}
```

Fig. 10.12. Select de les dades que es volen mostrar a la taula.

Per introduir les dades dins de la taula i poder mostrar-les, s'ha de fer ús d'una nova classe, aquesta és UITableViewCell. La classe UITableViewCell defineix els atributs i el comportament de les cel·les que apareixen en els objectes UITableView.

Aquesta classe, ens permet anar afegint un per un, tots els resultats de la nostra consulta dins de la taula que es mostra per pantalla.

```
- (UITableViewCell *)tableView:(UITableView *)tblView
cellForRowAtIndexPath:(NSIndexPath *)indexPath {
        static NSString *CellIdentifier = @"Cell";

    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier];
    if (cell == nil) {
        cell = [[[UITableViewCell alloc] initWithFrame:CGRectZero
reuseIdentifier:CellIdentifier] autorelease];
    }

        switch(indexPath.section) {
          case 0:
                    cell.text = clientObj.clientName;
                    break;
          case 1:
                    cell.text = [NSString stringWithFormat:@"C/ %@", clientObj.adreca];

                    break;
        }

        return cell;
}
```

Fig. 10.13. Assignació dels valors a les cel·les de la taula.

Al final de la taula, s'hi troben dos botons. El primer botó és "Veure ruta", el qual enviarà a una nova vista, la qual mostrarà un mapa amb la posició geogràfica de cadascun dels clients assignats, així com la ruta a seguir a traves d'ells. En el punt 10.2.1.1 s'explica aquest apartat amb més profunditat. El segon botó és "Menú principal", el qual retornarà al menú principal de l'aplicació.

Fig. 10.14. Ruta.

Aquesta vista permet també visualitzar detalladament les dades de cadascun dels clients. Al clicar la cel·la d'un client qualsevol, aquest ens enviarà a una nova vista amb les dades complertes del client en qüestió. Per poder visualitzar aquestes dades, s'ha tornat a fer ús del UITableViewController per a mostrar les dades i del UINavigationController per poder tornar a la vista anterior en qualsevol moment.

Quan passem a aquesta segona vista de taula, no cal que tornem a fer la consulta a la base de dades, ja que aquesta queda guardada en memòria, concretament en el "delegate".

La delegació és un model simple i de gran abast en el qual un objecte en un programa actua, depèn, o treballa amb un altre objecte. L'objecte de la delegació manté una referència a l'altre objecte (el delegat) i en el moment adequat envia un missatge a la mateixa. El missatge informa el delegat d'un esdeveniment que l'objecte de la delegació està a punt d'utilitzar o ja ha estat utilitzat. Això permet que en aquest cas, quan es clica en un client, es mostrin les dades referents a aquell client.

Fig. 10.15. Vista detallada del client.

10.2.1.1 Veure Ruta.

Aquesta funcionalitat de l'aplicació és, des de el punt de vista de codificació i investigació, la més complexa de totes. Es tracta de visualitzar la ruta a seguir per un treballador, traçada a través de les adreces de tots els clients que es tenen assignats.

Iphone treballa amb la API de Google Maps per poder mostrar mapes per pantalla. Aquesta API ofereix múltiples possibilitats, entre les quals estan les que s'han usat per realitzar aquesta aplicació.

Aquestes possibilitats són:

- Mostrar un mapa detallat del món sencer.
- Establir anotacions (també anomenats pins o punts) que assenyalen adreces o coordenades concretes.
- Mostrar la posició actual del dispositiu mòbil.
- Dibuixar rutes entre els punts prèviament establerts.

A continuació s'expliquen detalladament els diferents punts.

Mostrar un mapa detallat del món sencer

El primer pas per a poder mostrar un mapa amb una ruta dibuixada entre diferents punts, és mostrar el mapa.

La classe necessària per a poder mostrar el mapa és MKMapView. La classe MKMapView proporciona una interfície de mapa integrable, similar a la proporcionada per l'aplicació Google Maps. Aquesta classe s'utilitza per mostrar la informació del mapa i manipular el seu contingut.

Però aplicant aquesta classe, encara no es pot mostrar el mapa per pantalla, ja que es necessita que aquest faci servir un framework creat per Apple. Aquest framework és MapKit. El framework MapKit proporciona una interfície per integrar els mapes directament en les finestres i vistes pròpies de l'aplicació.

Aquest marc també proporciona suport per crear anotacions en el mapa, afegir superposicions, i realitzar cerques de geo-codificació (coordenades).

```
@interface MapView : UIView<MKMapView>
```

Fig. 10.16. Declaració d'una vista que mostrarà un mapa.

Per poder mostrar el mapa, la vista ha de afegir una sub-vista, aquesta és la que mostrarà el mapa.

```
mapView = [[MKMapView alloc] initWithFrame:CGRectMake(0, 0, frame.size.width,
frame.size.height)];
[mapView setDelegate:self]; [self addSubview:mapView];
```

Fig. 10.17. Afegir sub-vista que mostrarà el mapa.

Si s'afegeix una vista d'un mapa, sense interactuar amb ell, la imatge que veuriem per pantalla seria semblant a la de la figura 10.18.

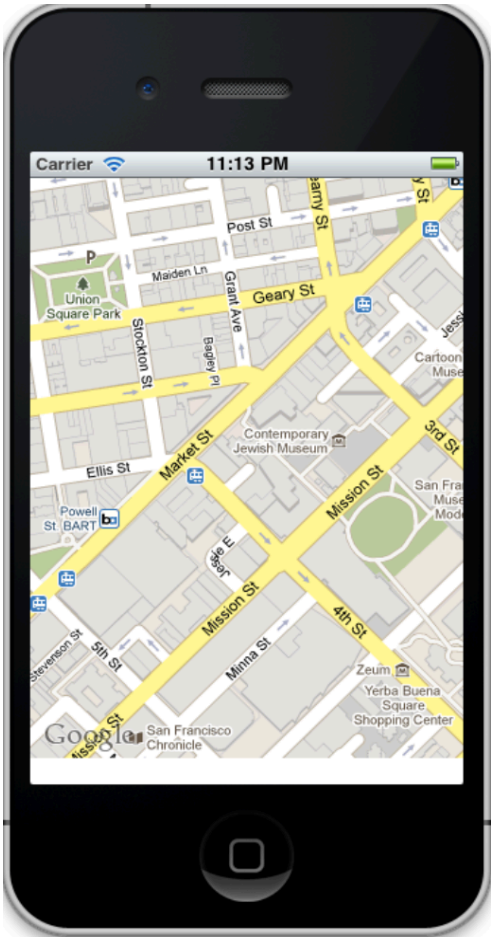


Fig. 10.18. Vista del mapa.

Establir anotacions

Una de les funcionalitats que té la API de Google Maps, és poder buscar adreces i mostrar-les al mapa. La manera que té de mostrar-les és col·locant una anotació en forma de xinxeta al mapa. Aquesta anotació mostra el lloc exacte que s'ha passat al mapa, a la vegada, si es fa clic a aquesta anotació, hi ha la possibilitat de mostrar informació pertinent a la adreça, ja sigui un nom, unes coordenades, o el que el desenvolupador necessiti mostrar.

Per poder mostrar aquestes anotacions, es necessita usar la classe MKAnnotationView. La classe MKAnnotationView és responsable de presentar visualment anotacions en una vista de mapa. Aquestes anotacions estan acoblades a un objecte d'anotació corresponent, que és un objecte que es correspon amb el protocol MKAnnotation. El protocol MKAnnotation s'utilitza per proporcionar informació relacionada amb l'anotació a una vista de mapa. Per utilitzar aquest protocol, cal que s'adaptin tots els objectes a mesura que s'emmagatzemen o representen dades de l'anotació. Cada objecte llavors serveix com a font d'informació sobre un mapa d'una sola anotació i proporciona informació crítica, com ara la ubicació de l'anotació al mapa.

A la vegada que s'usa aquesta classe, també s'ha d'afegir un framework creat per Apple, que es l'encarregat de detectar les coordenades actual de l'Iphone, això permetra que també es mostri una anotació de la nostra posició actual, cosa que serà molt útil en aquesta aplicació, ja que permetrà mostrar la ruta que està seguint en tot moment el treballador. Aquest framework és Core Location.

```
@interface PlaceMark : NSObject <MKAnnotation> {
        CLLocationCoordinate2D coordinate;
        Place* place;
}
```

Fig. 10.19. Utilització de la classe MKAnnotation i aplicació del framework CoreLocation.

La manera de poder mostrar aquestes anotacions dins el mapa a partir d'unes direccions és la següent:

- Es llegeixen les direccions de la base de dades o les introduïdes per l'usuari.
- S'envien a Google per que ens retorni les coordenades (latitud i longitud)
- Un cop rebudes es mostren per pantalla.

Per llegir les adreces que es volen fer servir, és tan senzill com usar l'objecte de la classe que s'usava per mostrar-les per pantalla a la taula. Com s'ha comentat anteriorment, aquestes dades es guarden en el delegate de l'aplicació, cosa que permet fer ús d'elles també en aquest moment. D'aquesta manera només s'ha de passar la adreça a la API de Google Maps, de manera que ens retorni aquestes coordenades. Per que Google ens les retorni, s'han d'enviar d'una manera concreta, en la figura 10.20 es pot observar com.

```
NSString *urlString = [NSString
stringWithFormat:@"http://maps.google.com/maps/geo?q=%@&output=csv",
[clientObj.adrecastringByAddingPercentEscapesUsingEncoding:NSUTF8StringEncoding]]
```

Fig.10.20. Enviar direccions a Google Maps.

El que es fa en aquesta línia de codi, bàsicament, és recollir la direcció que està guardada al delegate i enviarla a la API de Google Maps. Google retorna un arxiu en format .CSV que l'aplicació guarda en un NSString que més endavant s'usarà per mostrar l'anotació. S'ha de puntualitzar que per enviar aquesta direcció, s'ha de forçar a l'aplicació que l'enviï en un format concret, això és el que fa la línia NSUTF8StringEncoding.

Quan es tenen les dades que ha retornat Google, el que s'ha de fer, es guardar la latitud i la longitud de la direcció que s'ha enviat prèviament, això es pot veure a la figura 10.21.

```
NSArray *listItems = [locationString componentsSeparatedByString:@","];

    if([listItems count] >= 4 && [[listItems objectAtIndex:0] isEqualToString:@"200"]) {
      latitud = [[listItems objectAtIndex:2] doubleValue];
      longitud = [[listItems objectAtIndex:3] doubleValue];
}
```

Fig. 10.21. Recollida de la latitud i la longitud.

Com es pot observar, es dóna l'ordre a l'aplicació que agafi tots els components, aquests components estan separats per comes, de manera que l'aplicació sap que cada dada esta entre comes. Cal remarcar que Google no retorna només les dades longitud i latitud, sinó que també incorpora algunes dades més, les quals no es necessiten, però que si s'usen per poder identificar si el resultat de la consulta a Google és correcta o no. Com es pot observar a la figura 10.22, Google ens retorna 4 dades, la primera diu que la recerca és

correcta, la segona és de control però no s'usa, i la tercera i la quarta són la latitud i la longitud per aquest ordre.

```
200,4,41.3879170,2.1699187
```

Fig. 10.22. Consulta a Google Maps.

La manera de recollir la latitud i la longitud, es pot veure a l'anterior figura 10.21, en la que, si el resultat de la consulta comença per 200 (consulta correcta), guardem la latitud i la longitud, per fer-ho s'ha d'especificar en quina posició del resultat estan, en aquest cas es la tercera i quarta posició, però si tenim en compte que la primera posició és considerada la zero, aleshores per recollir la latitud s'ha d'especificar que està situada a la posició 2 del resultat, i la longitud està situada a la posició 3 del resultat. Per poder especificar-ho s'usa objectAtIndex, funció que permet accedir a una posició concreta d'un Array.

Un cop es tenen les coordenades guardades, només queda mostrar-les al mapa. Per fer això s'ha creat una classe anomenada Place. La qual té declarades 4 variables, un nom, una direcció, una latitud i una longitud. D'aquesta manera es pot aconseguir mostrar tantes anotacions com es desitgi, ja que tan sols s'han de crear objectes de la classe Place, i enviar-los al mapa. El que es fa es enviar un objecte de la classe Place cap al protocol MKAnnotation, i aquest s'encarrega de mostrar-lo a la vista, que en aquest cas es MKAnnotationView. En la figura 10.23 es pot observar com es crida a la funció addAnnotation, la qual pertany al protocol MKAnnotation, i es passa el Place amb nom from, i s'especifica a quina vista es vol mostrar, en aquest cas es mapView que està declarat com un MKAnnotationView.

```
[mapView addAnnotation:from];
```

Fig. 10.23. Enviem un Place al protocol MKAnnotation.

Un cop enviades totes les anotacions que es volen mostrar, l'aplicació mostrarà per pantalla el mapa, i les anotacions que s'hagin enviat, d'una manera semblant a la figura 10.24.

Fig. 10.24. Vista d'un mapa amb anotacions.

Mostrar la posició actual del dispositiu mòbil

La tercera funcionalitat que s'usa en aquesta aplicació, és la de mostrar la posició en tot moment del dispositiu mòbil. Això permetrà que el treballador pugui veure en tot moment on està i cap a on va. Es podria dir que fa ús d'un GPS.

Per poder mostrar la posició actual, tant sols s'ha de comunicar a la vista MKMapView que la mostri, això s'aconsegueix amb una simple línia de codi.

```
mapView.showsUserLocation = YES;
```

Fig. 10.25. Mostrar posició actual del dispositiu.

La funció showsUserLocation, permet que l'aplicació reconegui la posició del dispositiu, aquesta posició es enviada a la vista del mapa prèviament declarada. Un cop enviada el mapa la mostra en forma de bola de color blau com es pot veure a la figura 10.26. La funció showsUserLocation pertany al framework CoreLocation, el qual s'ha implementat prèviament a l'aplicació.

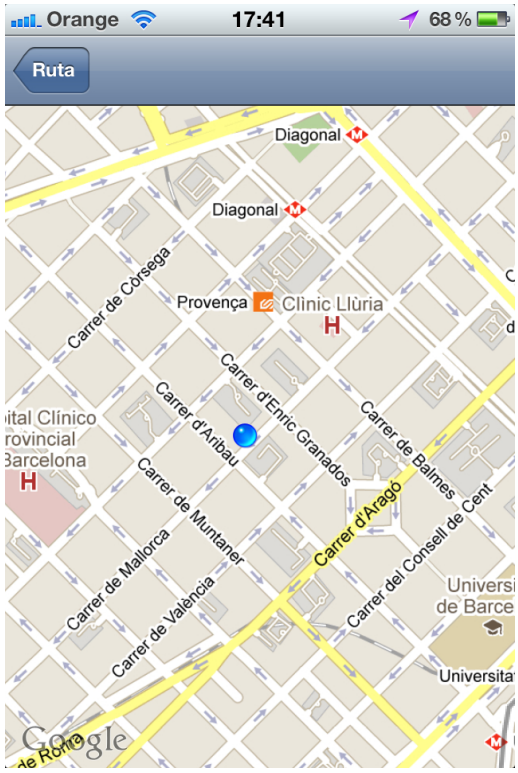


Fig. 10.26. Posició actual del dispositiu.

Com es pot observar també en la figura 10.26, a la part superior dreta de la pantalla, hi ha una fletxa, la qual indica que s'està fent ús del GPS del Iphone, el qual s'activa a partir de la funcion showsUserLocation, el qual fa la crida GPS per obtenir les coordenades del dispositiu.

Dibuixar rutes

La última funcionalitat usada en mapes, és la de dibuixar rutes entre anotacions. Aquesta és la part més complexa de totes, ja que a part de fer ús de totes les funcions i frameworks que s'han usat fins ara, n'afegeix de nous, els quals són més complicats de gestionar. A més s'han d'afegir alguns inconvenients que s'han trobat a mida que s'anava avançant el projecte.

Google Maps permet la creació de rutes entre diversos punts, el problema es que si aquesta ruta es vol fer desde Iphone, el resultat obtingut no es el mateix. Si s'aplica la API de Google Maps tal i com s'ha d'aplicar, el resultat obtingut no és el desitjat. En aquest projecte es vol mostrar la ruta entre diversos punts, que van des d'un mínim de 2, fins a un màxim determinat per el nombre de clients que es tinguin assignats. Si aquesta ruta es fes directament a la web de Google Maps, aquesta ens mostraria el resultat desitjat, però al fer-la des de Iphone, el resultat obtingut és que mostra totes les anotacions enviades, però només dibuixa la ruta entre les 2 últimes anotacions, provocant que no veiem la ruta total. Aquest problema es pot comprovar en l'aplicació nativa que incorpora l'Iphone, la qual usa Google Maps per cercar direccions o crear rutes, la qual només permet avui dia crear rutes entre 2 posicions. Això ve donat per problemes de comunicació entre Google Maps i Iphone, que de moment no tenen solució, a no ser, que "s'enganyi" al Iphone.

Per obtenir una ruta creada per Google Maps i mostrar-la, l'aplicació ha de fer una crida a la API concreta, la qual es pot veure a la figura 10.27.

```
NSString* apiUrlStr = [NSString
stringWithFormat:@"http://maps.google.com/maps?output=dragdir&saddr=%@&daddr=%
@", saddr, daddr];
```

Fig. 10.27. Consulta de Ruta a Google Maps.

En aquesta consulta s'envien 2 direccions, una que és la direcció d'inici i l'altre que és la direcció final. Google retorna les dades necessàries per poder dibuixar la ruta. En aquest cas s'ha fet la consulta per una ruta de 2 direccions, el problema ve quan enlloc de 2 s'envien 3 o més direccions, la API de Google Maps retorna un resultat buit, el qual no permet dibuixar la ruta correcta.

En la figura 10.28 es pot observar quin és el resultat de la consulta feta a Google Maps en el cas de sol·licitar una ruta entre 2 direccions.

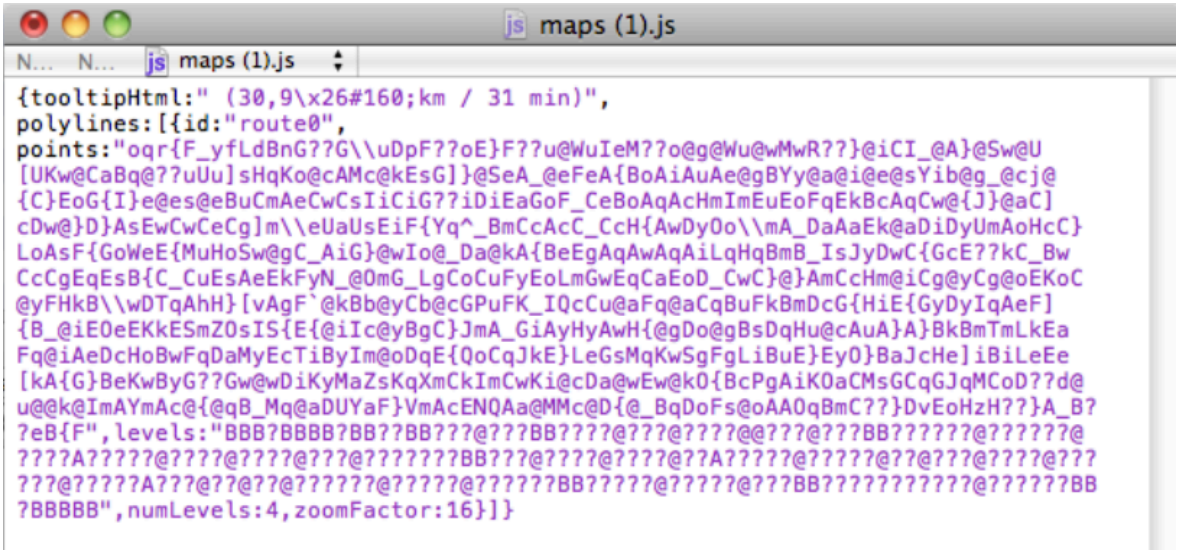


Fig. 10.28. Resultat de la consulta a Google Maps.

Aquest script, proporciona una sèrie de dades, que s'han de recollir i tractar de certa manera per a poder obtenir la ruta i dibuixar-la. En concret les dades de la ruta venen donades just després de la variable "points:". El que s'ha de fer, es forçar a l'aplicació que només llegeixi aquestes dades, les quals més endavant serviran per a poder dibuixar la ruta. La manera de fer-ho, és la que es pot observar a la figuta 10.29.

```
NSString* encodedPoints = [apiResponse stringByMatching:@"points:\\\"([^\\\"]*)\\\""
capture:1L];
```

Fig. 10.29. Recollir les indicacions de la ruta a seguir.

En aquesta línia de codi, s'està dient a l'aplicació, que quan trobi la paraula "points:" reculli tota la informació que ve a continuació, i la guardi en un NSString.

Però amb aquesta consulta a la API de Google Maps, s'ha de recordar que hi ha el problema de que només es pot obtenir la ruta entre dos punts, aleshores, s'ha de pensar de quina manera, podem mostrar la ruta entre tots els punts que l'aplicació requereixi.

La solució trobada ha sigut aprofitar que el delegate guarda en tot moment la informació que l'aplicació requereixi, així doncs, s'ha pensat en fer tantes consultes com N-1 direccions es tinguin. D'aquesta manera s'obtindran les rutes entre tots els punts

disponibles, i només caldrà dibuixar-les una a una, però aquí s'ha de tenir una cosa molt important en compte, fins ara sempre que s'enviaven dades a una vista, el que es feia era actualitzar la vista amb la nova informació, si es fa d'aquesta manera, es tindrà el mateix problema que es tenia des d'un inici, que només es visualitzarà per pantalla la última de les rutes, per tant, enlloc d'actualitzar s'ha d'afegir a la vista la nova ruta. Sembla una qüestió trivial, però el normal en la programació per Iphone, és actualitzar la vista, no afegir-hi noves dades, per això és important fer referència a aquesta particularitat.

Un cop es tenen guardades totes les rutes de les N-1 direccions de les que requereix l'aplicació, s'ha de dibuixar la ruta. Per fer això es fa ús novament del framework MapKit, el qual te la classe MKPolyLine. La classe MKPolyLine representa una funció que consisteix en un o més punts que defineixen la connexió de segments de línia. Els punts estan connectats d'extrem a extrem en l'ordre en que es proporcionen. Tenint aleshores un array en el qual hi consten totes les coordenades en les que l'aplicació hi haurà de dibuixar una línia per acabar formant una ruta. Es pot veure a la figura 10.30 com s'afegeixen una a una les coordenades pertinents a la ruta.

```
NSNumber *latitud = [[[NSNumber alloc] initWithFloat:lat * 1e-5] autorelease];
NSNumber *longitud = [[[NSNumber alloc] initWithFloat:lng * 1e-5] autorelease];
CLLocation *loc = [[[CLLocation alloc] initWithLatitude:[latitud floatValue]
longitude:[longitud floatValue]] autorelease];
[array addObject:loc];
```

Fig. 10.30. Creació del array amb les dades de la ruta.

En aquesta part de codi, es pot veure com s'obtenen una a una les coordenades (latitud i longitud) de tota la ruta, per a continuació guardar-les en un array. Abans de guardar-les s'han de convertir al format adequat per a que la funció encarregada de dibuixar la ruta pugui entendre-les, el format adequat el proporciona la classe CoreLocation.

Un cop es tenen totes les coordenades guardades, només queda un últim pas, dibuixar la ruta. Per dibuixar-la s'ha fet ús del framework CoreGraphics. El framework CoreGraphics proporciona dues classes molt útils en aquest cas, una és la classe UIColor, i l'altre és la classe UIIMage, les quals s'usen per a dibuixar la ruta i mostrar-la sobre el mapa. El que fa l'aplicació és crear un objecte de la classe UIColor per a cada coordenada de la ruta a seguir, per a continuació guardar-la en un objecte creat de la classe UIIMage, per a finalment afegir-ho a la vista de mapa creada al principi de l'aplicació. Amb això

s'aconsegueix que a la vista del mapa s'hi visualitzin les diferents anotacions, juntament amb la ruta dibuixada, la qual es dibuixa amb una línia del color que es desitgi, en aquest cas de color gris. Podem veure en la figura 10.31 com es realitzen tots els passos que s'acaben de descriure.

```
-(void) updateRouteView {
CGContextRef context = CGBitmapContextCreate(nil, rutaView.frame.size.width,
rutaView.frame.size.height, 8, 4 *
rutaView.frame.size.width,CGColorSpaceCreateDeviceRGB(),
kCGImageAlphaPremultipliedLast);
CGContextSetStrokeColorWithColor(context, liniaColor.CGColor);
CGContextSetRGBFillColor(context, 0.0, 0.0, 1.0, 1.0);
CGContextSetLineWidth(context, 3.0);
for(int i = 0; i < ruta.count; i++) {
        CLLocation* location = [ruta objectAtIndex:i];
        CGPoint point =
        [mapView convertCoordinate:location.coordinate toPointToView: rutaView];
        if(i == 0) {
                CGContextMoveToPoint(context, point.x, rutaView.frame.size.height -
point.y);
        } else {
                CGContextAddLineToPoint(context, point.x, rutaView.frame.size.height -
point.y);
        }}
      CGContextStrokePath(context);
      CGImageRef imatge = CGBitmapContextCreateImage(context);
      UIImage* img = [UIImage imageWithCGImage: imatge];
      rutaView. imatge = img;
      CGContextRelease(context);
}
```

Fig. 10.31. Dibuix de la ruta.

Un cop dibuixada la ruta, podem veure en la figura 10.32, com queda una ruta dibuixada amb les diferents anotacions corresponents a les direccions on es desitja anar, mostrant a més a més la posició actual de l'Iphone.

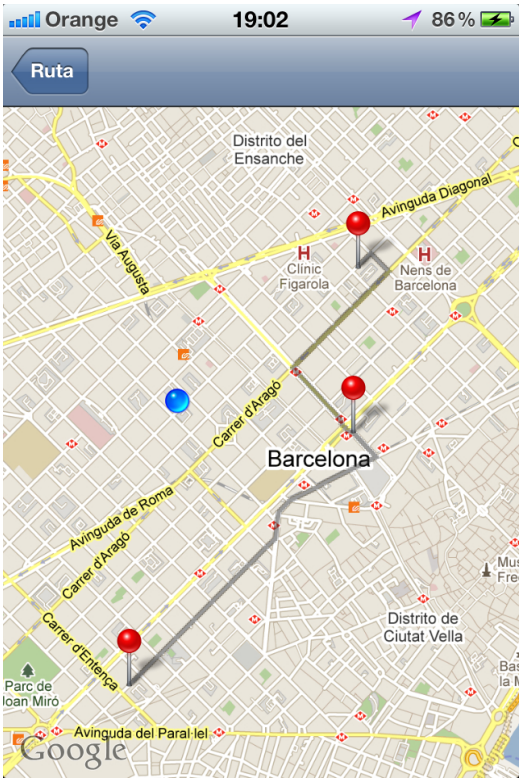


Fig. 10.32. Vista final del mapa amb totes les funcionalitats.

Per últim, cal destacar també que quan l'usuari desplaça el mapa d'un costat a l'altre, fa un zoom per apropar la pantalla o allunyar-la, s'han d'actualitzar les dades de la ruta, ja que sinó es fa, es veurien les anotacions, però la ruta que les uneix quedaria totalment descuadrada, per tant, s'ha de definir un mètode que sigui capaç de reconèixer aquestes variacions, i redibuixi la ruta. En la figura 10.33 podem veure el mètode regionDidChangeAnimated, el qual és l'encarregat de detectar aquestes variacions, un cop detectades, el que fa es tornar a executar la funció que s'ha mostrat en la figura 10.31 per redibuixar la ruta.

```
- (void)mapView:(MKMapView *)mapView regionDidChangeAnimated:(BOOL)animated
{
	[self updateRutaView];
	[self updateRutaView2];
	rutaView.hidden = NO;
	rutaView2.hidden = NO;
	[rutaView setNeedsDisplay];
	[rutaView2 setNeedsDisplay];
}
```

Fig. 10.33. Mètode regionDidChangeAnimated.

10.2.2. Mapa.

Aquest apartat afegeix una funcionalitat extra a l'aplicació. És molt semblant a l'apartat de veure ruta explicat en el punt anterior, però amb una petita variació. En l'apartat anterior el mapa mostrava per pantalla la ruta a seguir entre les diferents adreces dels clients assignats al treballador, però no permet alterar aquesta ruta, en canvi aquest apartat si que permet que el treballador interactuï amb el mapa.

El treballador quant accedeixi a l'apartat mapa fent clic al botó pertinent, visualitzarà a la pantalla diversos camps de text buits, i 2 botons, un per confirmar la introducció de dades i l'altre que dóna accés a l'apartat Ruta.

Es dóna a l'usuari l'opció d'introduir diverses adreces per tal de que si en algun moment ha de canviar la ruta per qualsevol motiu, ho pugui fer. Per posar un exemple clar, podria passar que el treballador rep una trucada de l'empresa en la que li comuniquen que ha d'anar a solucionar una incidència, aleshores el treballador pot introduir aquesta direcció al primer camp de text, i introduir altres direccions en els següents camps de text per trobar la ruta que ha de seguir, sempre s'han d'introduir un mínim de 2 direccions, ja que si només se n'introdueix una, la ruta no es pot crear.

Com a opció extra, s'ha afegit el botó "Veure Ruta" el qual permet veure al treballador la ruta assignada, per si vol fer servir alguna d'aquelles direccions en la nova ruta que ha de seguir, d'aquesta manera el treballador la pot consultar, per seguidament introduir-la al camp de text.

La figura 10.34 mostra la vista d'aquest apartat.

Fig. 10.34. Apartat Mapa.

10.2.3. Foto incidència.

El tercer apartat dels recursos del treballador, s'anomena "Foto incidència". En aquest apartat es dóna l'opció al treballador de realitzar fotografies en cas de que ho necessités, per exemple, s'hi es trobés amb alguna cosa trencada, o subjecte a revisió per part de l'empresa. D'aquesta manera fent una fotografia el treballador pot deixar constància d'aquesta incidència.

Quan el treballador clica el botó Foto incidència, l'aplicació l'envia a una nova pantalla on hi apareix un botó en el que hi apareix el text fer fotografia. Quan l'usuari prem aquest

botó automàticament s'obre la interfície de la càmera de fotografies de l'Iphone (figura 10.35).

Fig. 10.35. Interfície càmera.

Tot seguit, l'usuari només ha de fer la fotografia que desitja. Un cop feta l'aplicació dóna l'opció a l'usuari d'usar aquella fotografia prement el botó "Use" o fer-ne una altre prement el botó "Retake" per si ha sortit malament (figura 10.36).

Fig. 10.36. Acceptar fotografia.

Un cop el treballador ha decidit usar la imatge, l'aplicació la guarda a l'àlbum d'imatges de l'Iphone. L'usuari és informat per pantalla de que la imatge s'ha guardat correctament (figura 10.37).

Fig. 10.37. Imatge guardada correctament.

Per poder realitzar aquesta tasca, s'ha usat la classe UIImagePickerController. Aquesta classe és la que fa possible interactuar amb la càmera de l'Iphone. La classe UIImagePickerController gestiona interfícies personalitzables, subministrada per l'usuari del sistema per a la presa de fotografies i pel·lícules en dispositius Iphone que disposin de càmera, i per a l'elecció de les imatges guardades i pel lícules per al seu ús. En la figura 10.38 es pot observar la creació d'un objecte de la classe UIImagePickerController.

```
UIImagePickerController * picker = [[UIImagePickerController alloc] init];
```

Fig. 10.38. Objecte UIImagePickerController.

En la figura 10.39 es pot observar el codi complert que s'ha de fer servir per poder realitzar la seqüencia anteriorment explicada, obrir la càmera, fer la fotografia i seguidament guardar-la a l'àlbum d'imatges.

```
-(IBAction) getPhoto:(id) sender {
        UIImagePickerController * picker = [[UIImagePickerController alloc] init];
        picker.delegate = self;
        if((UIButton *) sender == escullFoto) {
          picker.sourceType = UIImagePickerControllerSourceTypeSavedPhotosAlbum;
        } else {
          picker.sourceType = UIImagePickerControllerSourceTypeCamera;}
        [self presentModalViewController:picker animated:YES];
}
- (void)image:(UIImage *)image didFinishSavingWithError:(NSError *)error
contextInfo:(void *)contextInfo
{
        UIAlertView *alert;
        if (error)
          alert = [[UIAlertView alloc] initWithTitle:@"Error" message:@"No s'ha pogut
guardar la foto." delegate:self cancelButtonTitle:@"Ok" otherButtonTitles:nil];
        else alert = [[UIAlertView alloc] initWithTitle:@"Guardat" message:@"Imatge
guardada a l'Àlbum de fotografies." delegate:self cancelButtonTitle:@"Ok"
otherButtonTitles:nil];
        [alert show]; [alert release];
}
```

Fig. 10.39. Codi guardar fotografia.

Es pot observar al principi un IBAction, el qual s'executa en el moment en el que es prem el botó per accedir a la càmera, en aquell moment es crea un objecte de la classe el qual la executa, seguidament, quan l'usuari accepta la fotografia i decideix guardar-la, s'executa l'script per guardar-la anomenat didFinishPickingMediaWithInfo, el qual s'encarrega de guardar la imatge a l'àlbum de fotografies. Finalment com a comprovació de que tot ha sortit correctament, s'executa la funció didFinishSavingWithError, la qual comprova que no hi ha hagut cap error durant el transcurs de la operació, i en cas d'error o d'èxit mostra un missatge per pantalla informant al treballador.

10.2.4. Enviar foto.

Aquesta és la última de les funcionalitats dels recursos del treballador. És el complement de la funcionalitat "Fer Foto", ja que a més de fer-la, amb aquesta nova funcionalitat es pot enviar la fotografia a un servidor per tal d'emmagatzemar-les.

Un cop el treballador ha realitzat la fotografia de la incidència, i aquesta ha quedat guardada a l'àlbum de fotografies de l'Iphone, el que ha de fer es prémer el botó "Enviar Foto". El treballador serà redirigit a una nova pantalla (veure figura 10.40) en la qual tindrà un menú a dalt de tot amb 2 botons.

Fig. 10.40. Menú enviar foto.

Com sempre, a dalt de tot, té l'opció de tornar al menú principal del treballador. Just a sota té els 2 botons abans comentats, el primer d'ells "Seleccionar", permet al treballador escollir una fotografia de l'àlbum de fotografies de l'Iphone. Quan l'usuari prem aquest botó automàticament es redirigit a la galeria d'imatges per escollir-ne una (figura 10.41).

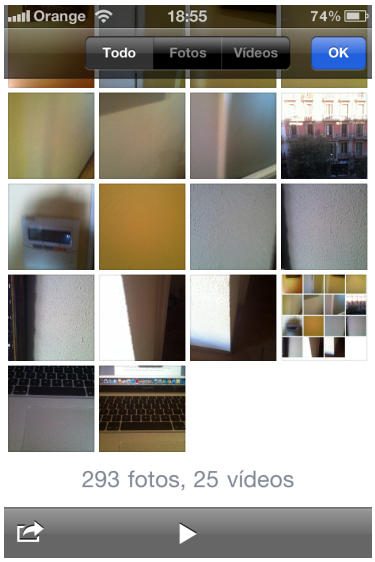


Fig. 10.41. Galeria d'imatges.

El treballador un cop l'ha seleccionat, veurà per pantalla la imatge seleccionada a pantalla completa (figura 10.42). Un cop fet això només queda enviar-la al servidor. Per enviar-la és tan senzill com prémer el botó "Enviar" situat a la part superior dreta, i al cap d'uns segons la imatge estarà allotjada al servidor.

Fig. 10.42. Enviar imatge.

Per realitzar aquesta funcionalitat, s'ha fet ús de Objective-C juntament amb PHP, per poder comunicar amb el servidor remot. Per codificar la part de l'aplicació d'Iphone, s'han usat les mateixes classes que per l'anterior (Fer Foto) ja que utilitzen el mateix, que és la interacció amb la càmera (la galeria d'imatges forma part de la classe UIImagePicker, en la que hi tenim la funció UIImagePickerControllerSourceTypePhotoLibrary la qual fa referència a la galeria), però aplicant unes petites directrius perquè en el moment donat en el que el treballador premi el botó Enviar, connecti amb el servidor, mitjançant un arxiu PHP i la guardi a la carpeta especificada en aquest arxiu PHP. Com es pot veure a la figura 10.43, al codi s'especifica la ruta exacte a la qual ha de connectar l'aplicació per a poder executar l'arxiu PHP que s'encarrega de connectar i guardar la imatge al servidor.

```
NSString *urlString = @"http://192.168.13.160:8888/enviar.php";
```

Fig. 10.43. Ruta on connecta amb l'arxiu PHP.

Es crea un NSString amb aquesta ruta, per posteriorment, connectar amb ella. La connexió es fa a partir del protocol POST, el qual PHP es capaç d'interpretar al moment de rebre informació. Per POST s'envia la imatge, i el nom que li volem posar a la imatge, en aquest cas s'ha usat "incidència" per fer referència a la fotografia. Com es pot veure a la figura 10.44, es crea un request per connectar amb el servidor, i posteriorment s'envien les dades de l'arxiu i la imatge a la direcció previament esmentada.

```
NSMutableURLRequest *request = [[[NSMutableURLRequest alloc] init] autorelease];
        [request setURL:[NSURL URLWithString:urlString]];
        [request setHTTPMethod:@"POST"];
[body appendData:[[NSString stringWithFormat:@"\r\n--%@\r\n", boundary]
            dataUsingEncoding:NSUTF8StringEncoding]];
[body appendData:[[NSString stringWithString:@"Content-Disposition: form-data;
name=\"userfile\"; filename=\"incidencia.jpg\"\r\n"]
            dataUsingEncoding:NSUTF8StringEncoding]];
[body appendData:[[NSString stringWithString:@"Content-Type: application/octet-
stream\r\n\r\n"]
           dataUsingEncoding:NSUTF8StringEncoding]];

[request setHTTPBody:body];
```

Fig. 10.44. Enviament de la imatge.

Una vegada s'executa aquesta senténcia, és quan entra en acció l'arxiu PHP. Aquest arxiu s'encarrega de rebre la petició feta per l'aplicació, recollir la imatge, i guardar-la. A la figura 10.45 es pot veure l'arxiu upload.php encarregat de dur a terme aquesta funció.

```
<?php
$uploaddir = './incidencies';
$file = basename($_FILES['userfile']['name']);
$uploadFile = $file;
$randomNumber = rand(0, 99999);
$date = strftime( "%d-%m-%Y-%H-%M-", time() );
$newName = $uploadDir . $date . $uploadFile;
if (is_uploaded_file($_FILES['userfile']['tmp_name'])) {
        echo "Arxiu enviat. \r\n";
} else {
        echo "Arxiu no enviat. \r\n";
}
if ($_FILES['userfile']['size']> 300000000) {
        exit("L'arxiu es massa gran.");
}
if (move_uploaded_file($_FILES['userfile']['tmp_name'], $newName)) {
   $postsize = ini_get('post_max_size');
   $canupload = ini_get('file_uploads');
   $tempdir = ini_get('upload_tmp_dir');
   $maxsize = ini_get('upload_max_filesize');
   echo "http://192.168.1.34:8888/{$file}" . "\r\n" . $_FILES['userfile']['size'] . "\r\n" .
$_FILES['userfile']['type'] ;
}
?>
```

Fig. 10.45. Upload.php.

Com es pot observar en la imatge anterior, rebem mitjançant la funció $_FILES els arxius amb el mètode POST. També es tenen diverses variables que serviran per poder guardar la imatge en el destí que seleccionem, aquesta funció la té la variable $uploaddir, la qual té assignada una carpeta allotjada a l'arrel del servidor anomenada "incidencies". A sota hi tenim la variable $date, la qual s'encarregara d'afegir la data i la hora actual al nom de la fotografia, així serà més senzill reconèixer la fotografia un cop allotjada al servidor. Abans

d'enviar la fotografia al servidor, s'estableix un control del tamany de la fotografia, en forma de bits, ja que si per error s'intentés enviar un arxiu de dimensions desproporcionades (arxius de vídeo per exemple), podria col·lapsar la xarxa del dispositiu mòbil. Un cop fet això, l'aplicació està en disposició d'executar la sentència per guardar l'arxiu, això es fa efectiu a l'última línia de codi, en el qual hi introdueix la ruta del servidor, seguida de la variable $file que conté les dades de la imatge així com la ruta exacte a la qual ha de ser guardada dins el servidor, o sigui la carpeta destí. Un cop la imatge ha estat guardada, l'usuari ja pot tornar a enviar una nova fotografia d'incidència o sortir i tornar al menú principal.

10.3. Panell principal – Gestió de clients.

La segona opció del panell inferior del menú principal, fa referència a la gestió dels clients. En ella s'hi pot veure el mateix menú superior i com no es manté el menú inferior que permet alternar entre els 2 menús de que disposa l'aplicació.

A la vista principal del menú de gestió de clients només hi ha una opció en forma de botó (figura 10.46), aquesta és la de crear un parte de treball. Quan el treballador arribi al client, i hagi realitzat tota la feina, haurà de crear un parte de treball, el qual servirà per deixar constància de la feina feta, i posteriorment per crear l'albarà i la factura pertinents.

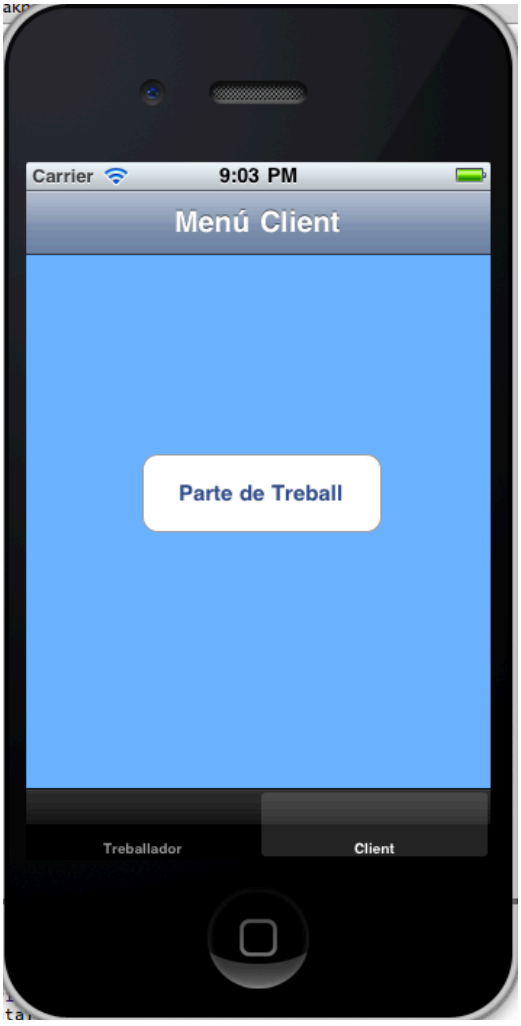


Fig. 10.46. Menú gestió de clients.

Per realitzat aquesta operació, s'ha usat un visor de navegador web. Per poder usar aquest navegador, s'ha usat la classe UIWebView. La classe UIWebView permet integrar contingut web en una aplicació. Per això, només cal crear un objecte UIWebView, adjuntar-lo a una finestra, i aquest enviarà una sol licitud per a carregar el contingut web. També es pot utilitzar aquesta classe per a moure's cap enrere i cap endavant en la història de les pàgines web, i fins i tot es poden establir algunes propietats de contingut Web mitjançant programació. S'utilitza el loadRequest: mètode per començar a carregar el contingut web. Com es pot veure en la figura 10.47, només cal indicar la ruta exacta de la web que es vol mostrar, i utilitzar el mètode LoadRequest per a executar-la.

```
- (void)viewDidLoad {
        NSURL *url = [NSURL
        URLWithString:@"http://192.168.13.160:8888/provaparte.php"];
        NSURLRequest *request = [NSURLRequest requestWithURL:url];
        [webView loadRequest:request];
        [[self webView] setDelegate:self];
        [super viewDidLoad];
}
```

Fig. 10.47. Càrrega de contingut web en una vista.

Pel que fa al contingut web, s'ha optat per utilitzar PHP. Aquest codi permet la interacció amb bases de dades, que és el que es necessita, a la vegada que és el més usat per aquest tipus d'aplicacions.

Per poder mostrar el contingut web d'una manera correcta, s'han usat les fulles d'estil CSS, amb elles es possible adaptar qualsevol tipus de web per a que es vegi correctament en un dispositiu mòbil. S'ha intentat que tot es visualitzi dins el marc de la pantalla de l'Iphone, de manera que la interacció del treballador amb la web sigui senzilla i ràpida. Això s'aconsegueix mitjançant diversos DIV els qual són molt útils a l'hora d'editar aquest tipus d'aplicacions per a mòbils. En la figura 10.48 es pot veure un exemple d'estil aplicat a un DIV.

```
#Layer1 {position:absolute;
left:10px;
top:16px;
width:100%;
height:560px;
}
```

Fig. 10.48. Estils CSS aplicats a DIV.

Al especificar que la posició és absoluta (position:absolute), s'està dient al DIV que s'ajusti a les necessitats de la pantalla per la qual s'està mostrant, això s'aconsegueix al aplicar el "width:100%", ja que es provoca que sempre ocupi el 100% de la pantalla. Si enlloc de visualitzar-ho en un Iphone ho visualitzéssim en un Ipad o en un tablet de majors dimensions, sempre es mostrarà en un tamany adaptat al dispositiu en qüestió.

10.3.1. Crear parte de treball.

El primer que haurà de fer el treballador un cop hagi finalitzat la feina al domicili del client, serà crear un parte de treball. En aquest parte hi figuraran les següents dades:

- Nom del client.
- Hores realitzades.
- Data.

El nom del client és una dada que es selecciona d'una llista creada en la que es mostren tots els noms dels clients. El treballador haurà de seleccionar el nom del client que acaba de realitzar.

Les hores realitzades són el total d'hores emprades per el treballador per a realitzar la feina pertinent.

La data com el seu propi nom indica, és la data en la que s'ha realitzat la feina.

Quan el treballador selecciona el nom del client al qual procedeix a crear el parte de treball, automàticament es mostra per pantalla un resum de la feina realitzada, la qual es carrega automàticament des de la base de dades (figura 10.49). Això es d'aquesta manera per que com es tracta d'un servei de manteniment contra incendis, el client sempre tindrà els mateixos productes (extintors, sprinklers...) per tant no cal afegir-ne de nous. Per mostrar aquestes dades s'ha fet ús de JavaScript, de manera que només es mostrin productes quan un client està seleccionat. Mitjançant un DIV amb la propietat onChange="showDiv(this.value);" es provoca que succeeixi d'aquesta manera (figura 10.50).

```
<?php mysql_select_db($database_pfc, $pfc);
$query_selectclient = "SELECT * FROM client";
$selectclient = mysql_query($query_selectclient, $pfc) or die(mysql_error());
$row_selectclient = mysql_fetch_assoc($selectclient);
$totalRows_selectclient = mysql_num_rows($selectclient);
?>
        <div style=" width:100%; height:30px;">
                <div style="float:left; font-size:12px; padding-left:20px; padding-
top:3px;"><label>Selecciona Client</label></div>
                <div style="float:right; font-size:12px; padding-right:20px;">
                        <select name="client" onChange="showDiv(this.value);" style="font-
size:12px; width:165px;">
                  <option value="">Escull...</option>
<?php
do {
?>
<option value="<?php echo $row_selectclient['id']?>" id="client<?php echo
$row_selectclient["id"];?>"><?php echo $row_selectclient['nom']?></option>
<?php
} while ($row_selectclient = mysql_fetch_assoc($selectclient));
  $rows = mysql_num_rows($selectclient);
  if($rows > 0) {
      mysql_data_seek($selectclient, 0);
          $row_selectclient = mysql_fetch_assoc($selectclient);
  }
?>
                </select>
                </div>
</div>
```

Fig. 10.49. Carrega de dades a un seleccionable.

```
<script type="text/javascript"><!--
var lastDiv = "";
function showDiv(divName) {
        // amagar l'ultim DIV
        if (lastDiv) {
                document.getElementById(lastDiv).className = "hiddenDiv";
        }
        // si el valor de la caja es null i hi ha un objecte amb aquest nom, a continuació,
cambiar la classe
        if (divName && document.getElementById(divName)) {
                document.getElementById(divName).className = "visibleDiv";
                lastDiv = divName;
        }
}
</script>
```

Fig. 10.50. Script per mostrar el DIV.

Un cop el treballador ha introduït totes les dades necessàries per a crear el parte de treball, només queda que premi el botó enviar, per a guardar el parte i navegar fins a la següent vista que serà la de l'albarà. Les dades s'envien mitjançant el metode POST, d'aquesta manera es fa possible que quan es passi a crear l'albarà, aquest ja tingui les dades introduïdes prèviament per el treballador.

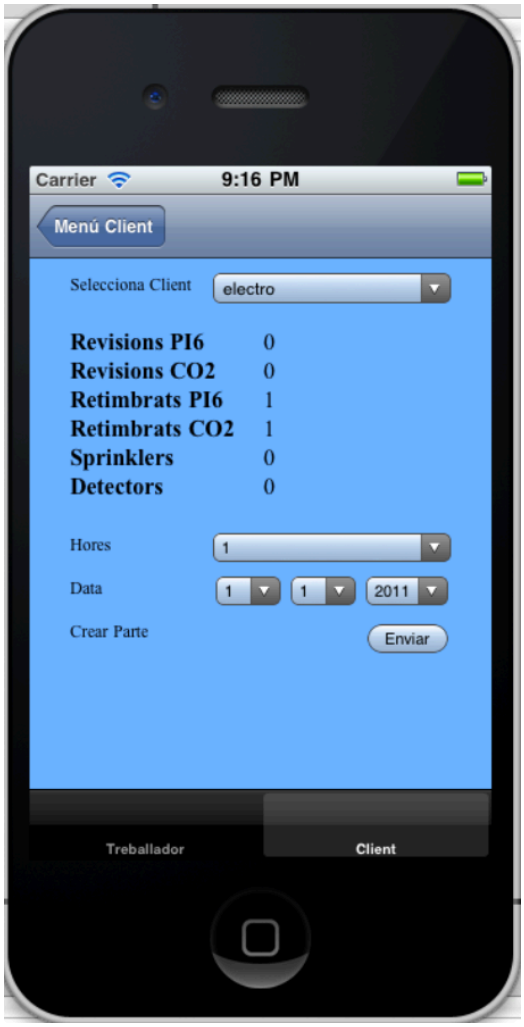


Fig. 10.51. Crear Parte.

10.3.2. Crear albarà.

Un cop el treballador ha creat un parte, automàticament és redirigit a la pantalla de crear un albarà. Aquesta pantalla mostra al treballador l'albarà generat en base al parte creat prèviament. En ell s'hi mostren les dades personals de l'empresa mantenidora i les del client, just a sota hi apareixen el número d'albarà, la data i el número de client. Tot seguit com es pot veure a la figura 10.52, es mostren les dades de l'albarà, o sigui un resum detallat de la feina realitzada, en el que hi consta el ID del producte, el nom i el seu preu unitari, després hi apareixen les hores dedicades amb el seu preu per hora, i al final el preu total sense IVA.

Fig. 10.52. Crear albarà.

Per poder mostrar les dades que es reben a partir del parte de treball, s'utilitza el mètode POST comentat anteriorment, en la figura 10.53 es pot veure com s'ha realitzat.

```
$idclient = $_POST['client'];
$hores=$_POST['hores'];
$dia =$_POST['dia'];
$mes =$_POST['mes'];
$any =$_POST['any'];
```

Fig. 10.53. Rebuda de dades amb el mètode POST.

Un cop el treballador ha revisat que les dades són correctes, procedirà a mostrar al client l'albarà en qüestió, per a la seva corresponent aprovació. Per que el client aprovi l'albarà i d'aquesta manera es pugui crear una factura, es requerirà que el client introdueixi una contrasenya de confirmació, per verificar que el client hi està d'acord (veure figura 10.54). Aquesta contrasenya serà proporcionada a cada client quan es doni d'alta al servei de manteniment contra incendis, el client serà informat de que cada vegada que es realitzi el manteniment, haurà d'usar-la per verificar les dades. Un cop el client hagi introduït la contrasenya, el treballador premerà el botó d'acceptar albarà per a confirmar l'albarà, guardar-lo i crear una factura. En cas de que la contrasenya sigui errònia es notificarà per pantalla aquest fet, i es donarà la opció al client de tornar a introduïr-la.

Fig. 10.54. Contrasenya del client.

10.3.3. Crear factura.

L'últim pas que s'haurà de realitzar és el de crear la factura. El treballador serà redirigit a aquesta pantalla un cop s'hagi acceptat l'albarà amb una contrasenya correcta. En aquesta factura hi apareixeran les mateixes dades que en el albarà, amb un parell de modificacions. Aquestes són que enlloc de mostrar el número d'albarà es mostra el número de factura, i que al final de tot s'hi ha afegit el preu total amb IVA.

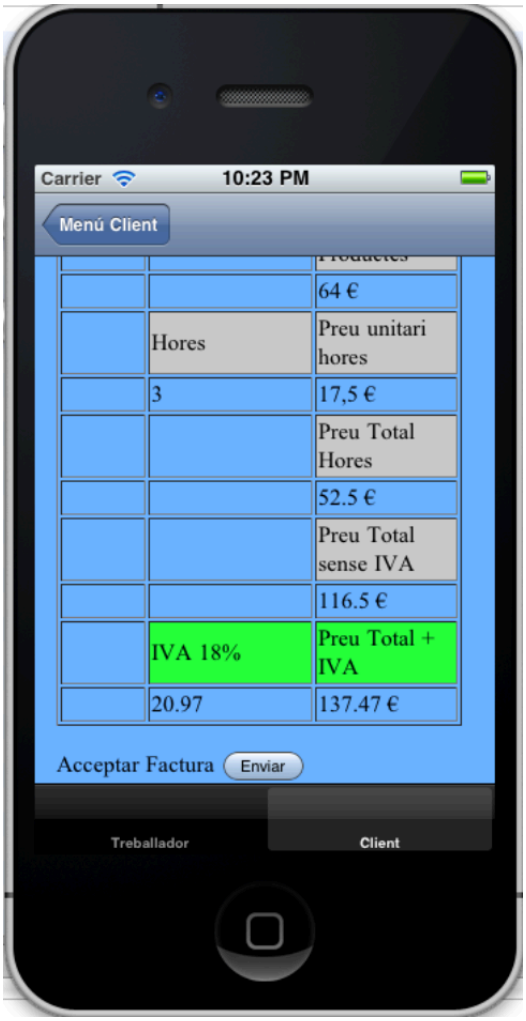


Fig. 10.55. Crear Factura.

De la mateixa manera que en l'apartat anterior, s'utilitza el mètode POST per a rebre totes les dades referents a l'albarà, les quals s'usen per a crear la factura.

Per acceptar la factura el treballador tan sols haurà de prémer el botó acceptar factura, per a guardar-la a la base de dades, un cop fet, es notificarà per pantalla al treballador que la operació ha estat realitzada amb èxit.

11. Conclusions.

La realització d'aquest projecte m'ha suposat moltes hores dedicades. Des d'un principi vaig haver de comprometre'm a aprendre un nou llenguatge de programació totalment desconegut per a mi.

Aquest projecte ha estat més complicat del que en un principi m'imaginava. El fet d'aprendre un nou codi de programació comportava una dificultat, però a mida que passaven les setmanes, cada cop trobava noves dificultats que alentien el correcte desenvolupament de l'aplicació. Això venia donat per l'escassa documentació sobre aquest codi, tan a Internet com a llibres especialitzats.

Tot i així, a títol personal, la realització d'aquest projecte final de carrera, m'ha permès ampliar els meus coneixements amb aquest nou codi de programació (Objective-C), i a sobreposar-me a grans dificultats.

M'he adonat, que a vegades gràcies a un petit consell d'un amic o company, pots trobar solucions a grans problemes, que després resulten ser de menor magnitud a la que semblaven. Un es dóna compte, de que tot, amb calma i paciència, es pot solucionar.

Per acabar, puc afirmar que estic content i orgullós d'haver acabat exitosament el projecte, un projecte que al principi era gairebé impensable per a mi, ja que suposava un repte molt gran.

12. Bibliografia.

Referències bibliogràfiques:

[1] Dr. Rory Lewis, Aplicaciones Iphone e Ipad para principiantes, Ed. Apress, 2009.

[2] Fernando López Hernández. El lenguaje Objective-C para programadores C++ y Java, MacProgramadores, 2008.

[3] Bert Altenburg, BecomeAnXcoder, Creative Commons Attribution, 2008.

[4] Javier Cala Uribe, Guía de desarrollo de aplicaciones móviles para iPhone / iPad, MaestrosdelWeb, 2009.

Localitzacions URL:

[1] www.maestrosdelweb.com/ , Tutorials Iphone.

[2] http://www.applenext.com/ , Manuals Objective-C.

[3] http://tutorialesiphone.wordpress.com/ , Tutorials aplicacions Iphone.

[4] http://blog.kadirpekel.com/ , Tutorials MapKit.

[5] http://www.programacioniphonesdk.com/ , Manuals CoreLocation.

[6] http://iphoneapp-dev.blogspot.com/ , Manuals Objective-C.

[7] http://www.zenbrains.com/ , Desenvolupadors aplicacions Iphone.

[8] http://blog.ujwaltrivedi.com/ , Tutorials Iphone.