Universitat Autònoma de Barcelona

escola d'enginyeria

(**9534-5: Actualització del software de Control de Presència**)

Memòria del Projecte Fi de Carrera
d'Enginyeria en Informàtica
realitzat per Enric Soler Rastrollo
i dirigit per Ramón Grau Sala
Bellaterra 5 de Juny de 2015

El sotasignat, ..... Ramon Grau Sala .....

professor/a de l'Escola d'Enginyeria de la UAB,

**CERTIFICA**:

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en/na ..... Enric Soler Rastrollo .....

I per a que consti firma la present.

Signat: .....

Bellaterra, ..... de ..... de 2015

# Agraïments

Vull agrair a tota la meva família pel suport que sempre m'han donat, els ànims rebuts i en especial als meus pares, per permetre'm la valuosíssima possibilitat de cursar uns estudis universitaris que m'agradaven.

A la May, pel seu suport, ànims i ajuda al llarg de tots els estudis i al llarg del projecte, suportant l'estrès de treballar, estudiar i fer el projecte.

Al Ramon, el director del meu projecte per tots els ànims rebuts, per acceptar la idea que li vaig proposar i les seves idees per treure el projecte endavant i fer un projecte interessant i per tota la paciència a respondre les consultes i llegir els diversos esborranys enviats.

Finalment, agrair a tots els companys i amics que he fet al llarg de tota la carrera, amb els que hem compartit bons moments, mals moments, moments amargs, però sobretot divertit al llarg d'aquests anys. Sobretot gràcies al Pou Team (Sergio, Raúl, Javier, Oscar, Alex, Julian, Manu, Xavi, Xescu...)

Contingut

Agraïments ..... 4
1 Introducció ..... 8
1.1 Objectius ..... 9
1.2 Estat de l'art: control d'accés i presència ..... 9
1.3 Planificació ..... 12
1.3.1 Planificació inicial ..... 12
1.3.2 Planificació final ..... 13
2 Fase prèvia: anàlisi de l'aplicació actual ..... 14
2.1 Empresa a estudiar: Enttia i Identtia ..... 14
2.2 Què és *IDPresence*? ..... 15
2.2.1 Terminal Manager ..... 16
2.3 Problemàtica ..... 17
3 Anàlisi de requeriments ..... 19
3.1 Anàlisi de l'aplicació actual ..... 19
3.1.1 IDPresence 2.0: aplicació anterior ..... 19
3.2 Anàlisi de l'aplicació futura ..... 19
3.2.1 IDPresence 4.0: objectius ..... 19
3.2.2 IDPresence 4.0: Nous requeriments ..... 21
4 Implementació ..... 22
4.1 Infraestructura: arquitectura client web – servidor ..... 22
4.2 Servidor ..... 23
4.2.1 Framework web: MCV.net ..... 23
4.2.2 ORM de Base de Dades: Entity Framework ..... 25
4.3 Client ..... 25
4.3.1 AngularJS ..... 25
5 Suport funcional ..... 30
5.1 Desenvolupament Iteratiu i incremental ..... 30
5.2 Principis SOLID ..... 30
5.3 Arquitectura 3-Tier ..... 31
5.3.1 Capa de dades ..... 31
5.3.2 Capa de negoci ..... 32
5.3.3 Capa d'aplicació ..... 32
5.3.4 Exemples: ..... 32
5.4 Entitats de l'aplicació ..... 33
5.4.1 Usuaris ..... 33

5.4.2 Credencials ..... 33
5.4.3 Seus, Edificis i Plantes ..... 34
5.4.4 Departaments ..... 35
5.4.5 Calendaris, tipus de dia i franges horàries ..... 35
5.4.6 Registres ..... 36
6 Beneficis de la migració tecnològica ..... 37
7 Test de qualitat ..... 38
7.1 Tests privats ..... 38
7.2 Tests a usuaris nous ..... 38
8 Conclusions ..... 40
9 Evolució futura ..... 42
10 Annex ..... 43
10.1 Glossari ..... 43
10.2 Webografia ..... 47
10.3 Taula de figures ..... 48
Resum ..... 49
Resumen ..... 49
Abstract ..... 49

# 1 Introducció

Una migració tecnològica és un tipus de projecte que consisteix en evolucionar un *software* existent a un altre (el mateix o un altre de diferent). Aquest canvi es pot dur a terme por molts motius diferents: rendiment de l'aplicació, obsolescència del programa, problemes de funcionament, manca de funcionalitat, etc. És una operació que s'ha de dur a terme amb molta cura i amb un anàlisi molt exhaustiu de la funcionalitat anterior i posterior per tal de no perdre tota la funcionalitat essencial i que ja es tenia.

És un procés seqüencial i que finalitza en un canvi de *software*. Les fases són les següents:

Primer de tot s'han d'analitzar els motius de la migració i els requeriments funcionals necessaris per a fer una transició correcte i per tal d'assegurar una migració satisfactòria. És molt important definir tots els motius i requeriments a priori per evitar caure en els mateixos problemes a mitja migració, ja que un error pot causar que el treball realitzat fins aleshores s'hagi de tornar a començar, s'hagi de modificar o fins i tot s'hagi de paralitzar la migració per problemes molt greus, cosa que implica costos molt elevats i fins i tot una cancel·lació del projecte.

Un cop ens assegurem que tots els requisits estaran reflectits en aquesta migració s'ha de decidir de quina manera s'implementarà la migració. Es pot decidir canviar el *software* o actualitzar l'aplicació actual. La decisió entre un i l'altre depèn del grau de satisfacció amb l'aplicació actual, i de la confiança que es tingui amb el programari. En el cas de seguir amb l'aplicació, es poden donar varis casos: que es faci un fent un canvi d'interfície, que es millori el rendiment, etc., coses que faran que es solucioni tota la problemàtica o es pot decidir fer una actualització complerta, des de 0 i utilitzant tecnologia nova i innovadora que doti al programa d'una nova característica atractiva: la modernitat. La decisió entre una actualització simple o un canvi de versió més complex, depèn de la complexitat de l'aplicació antiga, de la qualitat del codi, de la antiguitat, etc. Aquesta actualització no és una simple actualització de software si no que requereix un procés de adaptació del software anterior al software final i pot arribar a ser una versió totalment diferent a l'anterior.

El següent pas és decidir el procés de la migració. S'ha de decidir un procés d'actualització el menys traumàtic pels usuaris actuals de manera que no perdin funcionalitats que ja tenen i que el procés d'aprenentatge sigui el menor (sobretot en temps, ja que una adaptació lenta donarà lloc a un increment del cost per l'empresa) i per tal d'evitar la frustració dels usuaris actuals de l'aplicació. Per aconseguir això, es pot actuar amb un desenvolupament iteratiu i incremental que permeti a l'usuari acostumar-se al funcionament de la nova aplicació, donar feedback de les sensacions a l'hora d'utilitzar-la i anar controlant que el desenvolupament s'està duent a terme d'una manera correcte i sense problemes de base.

Per ajudar al procés de migració, es poden definir una sèrie de instal·lacions pilot que permetin obtenir un feedback de usuaris que no han treballat amb l'aplicació. Aquestes impressions poden ser bones per tal de fer millores d'usabilitat abans de estudiar fer una migració a tots els clients. Aquesta instal·lació serà paral·lela a la instal·lació anterior, es a dir, podran conviure si els usuaris ho creguessin convenient.

En aquest projecte, s'ha aplicat tot el procés de migració tecnològica analitzant la funcionalitat d'*IDPresence* (aplicació de control d'accés i presència de l'empresa Identtia) i s'ha desenvolupat una aplicació web que substitueix a l'aplicació d'escriptori original. Per tal de fer aquest procés, s'ha aplicat el mètode exposat anteriorment que ha constat de l'anàlisi de tota la problemàtica

de l'aplicació antiga i de tots els requeriments obligatoris que contenia més els nous. Seguidament, s'ha analitzat diferents solucions per tal donar respostes als problemes i aconseguir un *software* de qualitat. Finalment, s'ha implementat la solució escollida aconseguint el propòsit inicial; realitzar una versió sense els problemes amb els quals constava l'antiga versió i obtenir un *software* modern, consistent i de qualitat. A més a més, una vegada l'aplicació web ha evolucionat a una versió suficientment estable (després d'una sèrie de proves internes), s'ha implantat la solució beta a clients nous i existents per comprovar que l'aplicació donava un resultat satisfactori.

## 1.1 Objectius

L'objectiu principal del projecte és poder aplicar totes les fases de la migració tecnològica, de manera que es puguin dur a terme en el cas pràctic. D'aquesta manera s'arribarà a aconseguir una evolució del software que farà que s'aconsegueixi una versió millorada, que supleixi tots els problemes que tenia la versió original i que sigui atractiva i moderna en comparació amb les versions actuals del mercat.

Per altra banda, es vol tenir una visió general del procés de migració que ajudi a determinar i delimitar els passos a seguir per poder-ho aplicar a altres migracions tecnològiques.

## 1.2 Estat de l'art: control d'accés i presència

El control d'accés i de presència és un sector que creix dia a dia degut a la aparició de hardware cada cop més barat que permet gestionar dos aspectes importants de l'empresa: el compliment de horaris (hores extres, absentisme laboral, mitges jornades, etc.) i la restricció de zones dins d'un edifici; per exemple, evitar que un comercial baixi al magatzem molts cops per tal de controlar que les seves comandes surtin a temps entorpint la feina dels mossos de magatzem fins a la restricció d'usuaris corrents a l'entrada a una sala de servidors.

Fa uns anys, totes aquestes restriccions es feien molt manualment, amb sistemes de claus diferenciades, vigilància física amb personal extra, etc. Gràcies a la tecnologia, tot aquest sistema es pot automatitzar i assegurar la veracitat de informació gracies a la biometria.

Hi ha moltes empreses especialitzades amb aquest tipus de control, a nivell estatal i mundial, i que treballen amb el seu *software* i el seu hardware.

Una de les més conegudes a nivell mundial és Suprema Inc. Suprema és una companyia coreana que es va fundar al 2000. Tenen tota una sèrie de terminals que permeten connexions IP o RS-485 utilitzant un ampli ventall de credencials: credencials biomètriques (empremtes), targetes *RFID*, *passwords* o els últims models fins i tot lectura facial.


*Il·lustració 1 – Terminals del fabricant Suprema*

A la figura podem comprovar com hi ha terminals amb un sol tipus d'autenticació (el tercer funciona amb *RFID*), barrejant dos tipus (primer i segon utilitzen empremta i *RFID*) o més tipus (com l'últim que suporta empremta, *RFID*, *password* i lectura facial).

El problema de Suprema és el software. És un software de gestió molt bàsic que permet controlar tot els terminals i les seves restriccions, però és molt poc personalitzable i bastant limitat a les gestions de persones, no a l'explotació de les dades generades per aquestes persones.

Un altre fabricant de terminals que té un software de gestió de terminals exclusivament és Nitgen Inc. També és una empresa coreana que des de 1998 fabrica terminals i tecnologia relacionada amb la identificació.


*Il·lustració 2 – Terminals del fabricant Nitgen*

El punt fort d'aquestes empreses es el hardware, que proveeix una gran fiabilitat i uns *SDK* que permeten realitzar totes les operacions amb els terminals de manera senzilla i transparent a l'usuari.

A nivell nacional hi ha dues empreses molt importants que es dediquen al control d'accés i presència, a més del control de la producció.

El grup SPEC és una empresa espanyola dedicada al control d'accés i presència des de 2001. Són fabricants de hardware i de software de gestió de personal, control horari, accés, etc.

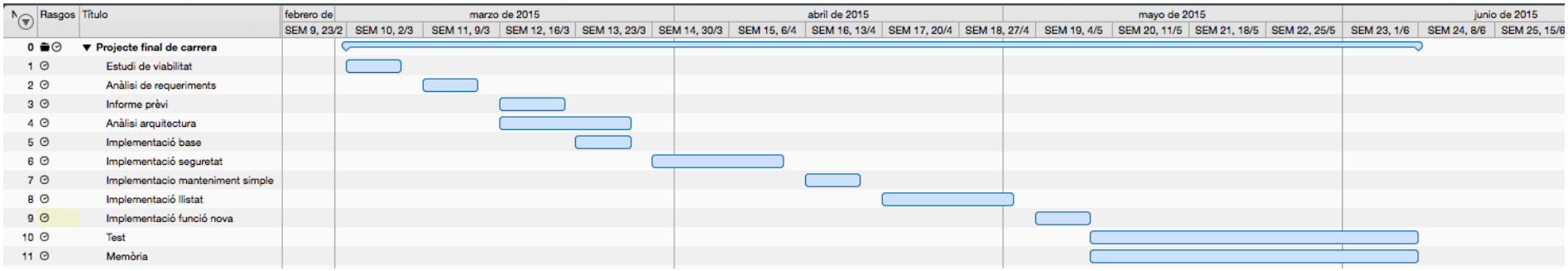
L'altre empresa important és Robotics. Robotics porta des de 1983 al sector i son fabricants de hardware i software.

El problema d'aquestes dues companyies és la complexitat del *software* que desenvolupen, majoritàriament a causa del control de producció. Són dues solucions que gestionen tot tipus de comportament: torns, quadres de demanda, accés, presencia, etc. Aquesta complexitat del *software* causa que alguns clients busquin alternatives més econòmiques i més senzilles d'ús i gestió, ja que a causa del control de producció s'afegeixen moltes opcions de configuració que dificulten l'ús de l'aplicació.

## 1.3 Planificació

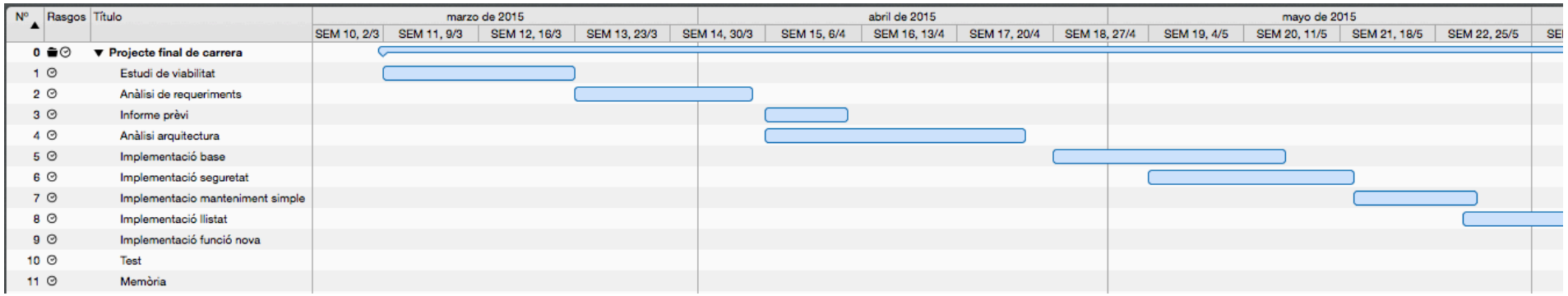
### 1.3.1 Planificació inicial

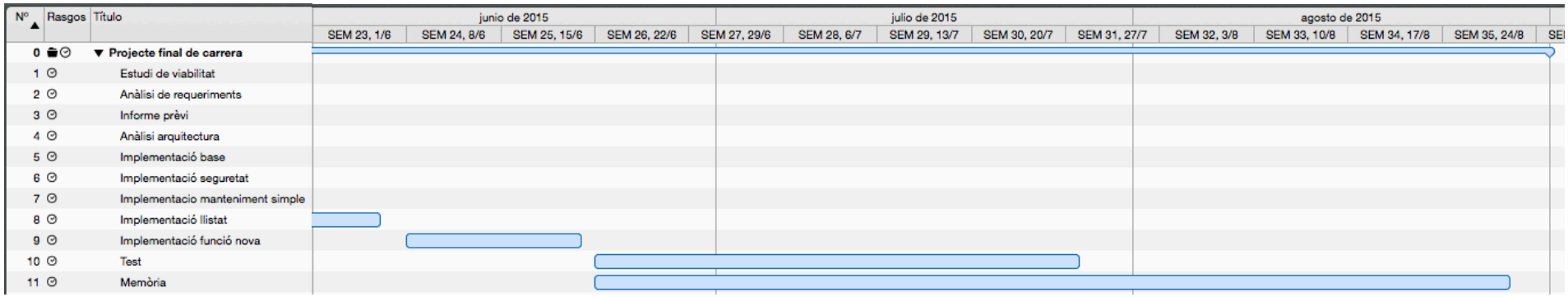
El projecte es va planificar de la següent manera:




Tot i haver planificat el projecte en 3 mesos, es va acabar allargant i no estava finalitzat el dia final previst. Aquesta planificació inicial va ser una proposta amb la qual va ser impossible fer compatibles les altres obligacions que tenia en el moment de començar. En una visió més realista, com és el que s'exposa en el següent diagrama cada etapa del treball es va ampliar d'una manera més extensa i amb un treball més exhaustiu.

### 1.3.2 Planificació final



# 2 Fase prèvia: anàlisi de l'aplicació actual

## 2.1 Empresa a estudiar: Enttia i Identtia

Enttia és una empresa dedicada a la enginyeria de Telecomunicacions, amb una naturalesa de negoci altament relacionada amb les tecnologies de la informació. Està especialitzada en projectes de telefonia, videoconferència, videovigilància, i control remot d'equipament i xarxes entre d'altres. Es considera una empresa d'enginyeria perquè intenta donar solució a tota la demanda de funcionalitat dins de l'àmbit de les telecomunicacions i el control remot de sistemes. A partir d' Enttia, i a causa de la demanda de *software* relacionat amb la identificació, sorgeix Identtia.

*Identtia* és una empresa especialitzada en el desenvolupament de software per controlar masses de gent: control d'aforament, d'esdeveniments, de presència, d'accés, etc. A més, Identtia es va especialitzar també amb la gran majoria de tecnologies utilitzades per identificar a gent, des de dispositius *RFID* (targetes de proximitat de 125 Mhz fins a les més modernes *Mifare Desfire*, passant per clauers o polseres *NFC*), dispositius biomètrics (captura d'empremtes o d'arbre de venes dactilars), etc.

El software d'Identtia es una bona solució per varis aspectes:

- Software de proximitat: es un software fet a Catalunya
- Contacte amb l'equip que ha realitzat el projecte: qualsevol problemàtica amb l'aplicació, passa per un primer filtre que evita que problemes de formació arribin als programadors, però si són problemes de funcionalitat es té un tracte molt proper client-programador que suposa una tranquil·litat pel client afegida.
- Suport tècnic: el suport tècnic el donen els propis programadors de l'aplicació. D'aquesta manera es pot explicar al client el perquè del funcionament i escoltar al client que fa suggerències per millorar-la.
- Integracions amb software de tercers: al tenir un tracte tant proper amb els usuaris, Identtia permet la integració de sistemes externs a les seves aplicacions. Això permet agafar un ventall més gran de clients que tenen software molt específic.

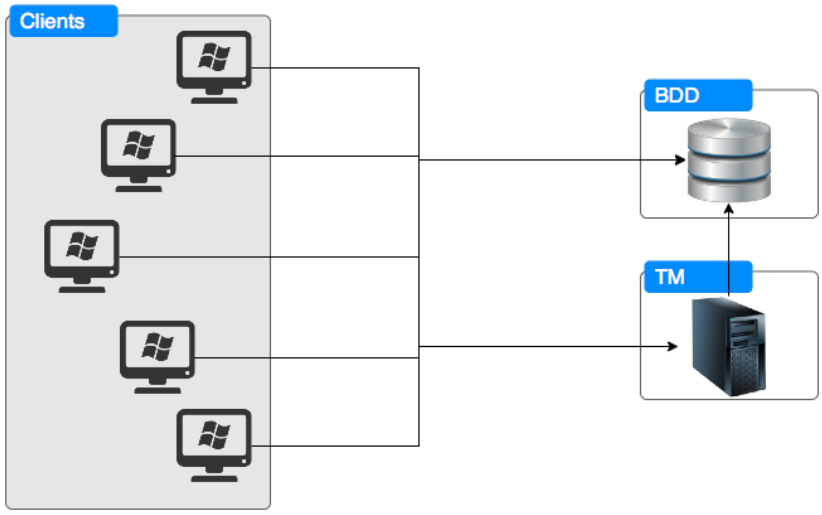
*Il·lustració 3 – Logos d' Enttia i Identtia*

## 2.2 Què és *IDPresence*?

IDPresence és el software de control de presència de la empresa Identtia. El nom sorgeix de la unió de **Id**entificació i de **Presence** o presència.

IDPresence és un software dedicat a la gestió del control de presència de persones. És una aplicació que permet conèixer en tot moment i d'una manera ràpida, eficaç i en temps real el control horari d'una sèrie d'usuaris. Gràcies a IDPresence es poden controlar la puntualitat dels empleats (entrada i sortida), el compliment dels usuaris (si un usuari compleix amb el seu horari, o fa hores extres o té absències remunerades o no), etc.

L'aplicació IDPresence és una aplicació que gestiona la presència d'empleats mitjançant una sèrie de terminals de diferents fabricants connectats a la xarxa. Als inicis, estava formada per dues parts: una aplicació que permetia mantenir les dades mestres, llistar dades, etc. i una altra aplicació que anava consultant els terminals per veure si hi havia marcatges.




*Il·lustració 4 – Infraestructura física IDPresence*

La primera versió va quedar-se obsoleta ràpidament per la dificultat de gestionar els problemes que hi havia amb l'aplicació que gestionava els terminals. Aquesta aplicació era robusta i depurada, però tenia problemes de tancament sobtats, sobretot per culpa del *multi-threading*. Aquests problemes de tancament i per tant, de sincronització de les dades dels terminals, feien que fos necessari desenvolupar una eina que controlés l'estat de l'aplicació de control (cosa que hagués fet que s'haguessin hagut de mantenir 3 aplicacions per separat).

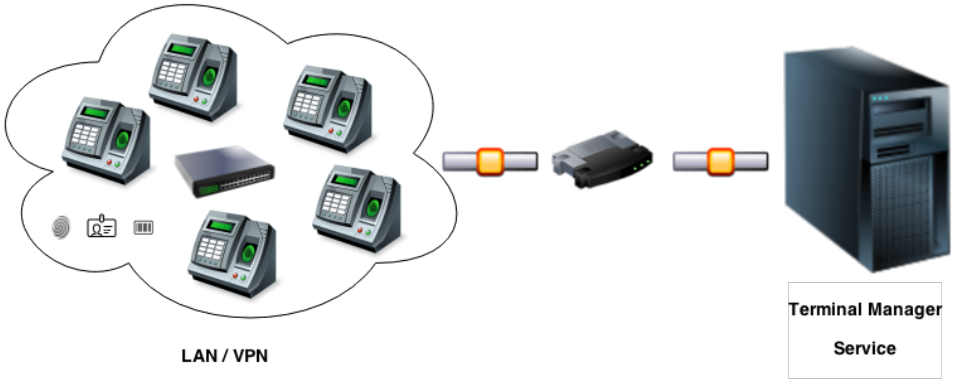
La següent versió (que és la versió actual) va sorgir a causa dels problemes amb la aplicació de control dels terminals. Per evitar aquests problemes, l'aplicació de control dels terminals va evolucionar cap a un servei del sistema operatiu. Aquest canvi comportava que la gestió de l'estat del servei fos gestionada automàticament pel sistema operatiu i a més implementava

noves funcionalitats: donava una sèrie de mètodes via *WCF* que permetien a IDPresence interactuar amb els terminals de manera remota mitjançant aquest servei. Aquests mètodes implementats al servei van acabar en una nova versió de IDPresence que comunicava amb el servei per tal de operar amb els terminals i creava el *Terminal Manager* que és com s'anomena el servei.

### 2.2.1 Terminal Manager

*Terminal Manager* és un servei de *Windows* dedicat a la interacció amb els terminals. És una part vital de tot el sistema de control d'Accés i de Presència.

Aquest servei s'encarrega de fer totes les operacions amb el terminal utilitzant normalment *SDK* de cada fabricant que es desitja implementar. Les operacions normals son les de sincronitzar les hores del servidor amb els terminals (importantíssim per si hi hagués algun problema amb diferències horàries entre terminals i servidor), recuperació de marcatges, registre d'usuari (amb les seves credencials i permisos), registre de grups d'accés, etc.




*Il·lustració 5 – Estructura física Terminal Manager*

El *Terminal Manager* s'alimenta de dades de la base de dades i de les dades dels terminals. Un cop recuperades les dades dels terminals, les guarda a la base de dades per tal de poder-les explotar amb la aplicació client.

Aquest servei seguirà sent l'actual, ja que com ja hem comentat, és molt més modern que el programa antic i ja funciona de forma correcta i estable. El nou programa s'haurà de comunicar amb aquest servei per tal de realitzar totes les operacions amb els terminals, per tant un requisit indispensable és que implementi la comunicació amb el *Terminal Manager* de manera correcte.

## 2.3 Problemàtica

Al llarg de tot aquest temps, la aplicació de presència ha quedat obsoleta en quant a interfície, funcionalitat, etc., ja que s'han centrat els esforços en generar el servei correctament i en altres projectes.

En el cas que ens interessa, la migració s'ha de dur a terme per varis problemes greus i que desemboquen en uns costos molt elevats (manteniment del *software*, càrrega incidental, etc.).

Per tal de solucionar això, es planteja una actualització de la aplicació a tots els nivells.

El problema més greu és la impossibilitat de desplegar la aplicació en ordinadors que contenen Sistemes Operatius de 64 bits, que son tots els sistemes operatius que venen actualment amb els nous ordinadors. La aplicació actual utilitza una sèrie de llibreries (propietàries) que no són compatibles amb els 64 bits i per tant s'han d'actualitzar. En alguns casos, fins i tot no ja no es possible perquè el fabricant ja no dóna suport a les llibreries o fan pagar per obtenir les llicencies.

Relacionat amb les llibreries de pagament, tenim un gran problema d'antiguitat de les llibreries de *UI*. Quan es va desenvolupar l'aplicació es va escollir una suite de llibreries enfocades a la interfície d'usuari que facilitaven la feina. Aquestes llibreries funcionen amb una modalitat de subscripció anual (de suport) i bianual (d'actualització de les versions contractades). Això afecta al cost general de l'aplicació, ja que s'ha de contractar una llicència par a cada desenvolupador. Aquestes llibreries són molt útils a l'hora de desenvolupar una aplicació, ja que afegeixen funcionalitat a les llibreries de Microsoft, i fan que el procés de desenvolupament s'acceleri, però tenen un problema, utilitzen una generació de codi que complica moltíssim el treball en paral·lel d'equips de programació, ja que generen molt codi diferent i fa que dos programadors diferents no puguin treballar alhora perquè quan es fa la pujada al Control de Versions hi ha conflictes molt difícils de resoldre.

Un altre problema és la infraestructura de la aplicació. La infraestructura dificulta el desplegament de la aplicació. Tal i com podem veure a la **¡Error! No se encuentra el origen de la referencia.**), una actualització completa de la aplicació, implica actualment 1 canvi al servidor del servei *Terminal Manager* i N actualitzacions a N ordinadors. En entorns poc restringits és un problema menor, ja que només es perd el temps de reinstal·lació i actualització de paràmetres de configuració que hagin pogut canviar d'una instal·lació a una altre. Però els problemes venen quan la instal·lació es fa en un entorn restringit, on els usuaris tenen permisos molt limitats. En aquests entorns, es depèn del suport de l'administrador de l'entorn a cada pas de la actualització, cosa que fa la actualització es torni molt més lenta, amb el corresponent cost de temps i diners. A més, actualitzar l'aplicació en un sistema amb molts usuaris es fa molt costós.

Un problema afegit és el manteniment de l'aplicació. En els inicis de l'aplicació es va intentar fer servir una arquitectura de capes que diferenciés l'accés al suport de dades separat de la interfície d'usuari. Aquest intent es va quedar aquí, ja que totes les connexions, cerques, etc., es fan des del mateix lloc: els formularis. Això fa que en un arxiu de codi es barregin tot tipus de funcions:

- Funcions de connexió amb la base de dades
- Comprovacions de regles de negoci
- Comprovacions de interacció amb l'usuari
- Tractament d'errors

Aquesta barreja de funcionalitat implica un cost alt de manteniment de l'aplicació, ja que localitzar el problema es complica a causa de la gran quantitat de línies de codi barrejades les unes amb les altres, a més de la necessitat de compilar tota la aplicació en comptes d'una llibreria en concret.

Finalment, l'últim motiu de la migració és la actualització de tota la interfície visual. L'aplicació té un disseny antiquat i la usabilitat de l'aplicació es força limitada a nivell de *UX*. A més, a nivell comercial els clients nous els costa veure una interfície d'aquest tipus en una aplicació nova per ells, ja que quasi en totes les aplicacions actuals, el component visual (a nivell de *UI* i *UX)* és molt important.


*Il·lustració 6 – IDPresence actual*

Al finalitzar amb aquest projecte, hauríem de tenir com a resultat una actualització de l'aplicació actual millorant els problemes que hi ha amb la versió anterior i alhora afegint funcionalitat al producte per fer-lo atractiu per a nous clients i tornar a captar els clients antics, que tenen una certa sensació de abandonament a causa de la falta d'actualitzacions del producte.

# 3 Anàlisi de requeriments

## 3.1 Anàlisi de l'aplicació actual

### 3.1.1 IDPresence 2.0: aplicació anterior

L'aplicació antiga, com hem pogut veure a la figura 4 té una estructura física de clients que connecten a una Base de Dades i a un servei per operar amb els terminals. Per a cada ordinador, s'instal·la un client que conté una configuració per instal·lació. Aquesta configuració conté paràmetres com tota la informació per connectat amb la Base de Dades on es connecta per recuperar la informació, la direcció IP del *Terminal Manager* i d'altres paràmetres necessaris més per a utilitzar l'aplicació. Aquesta configuració és per ordinador i no per usuari.

Aquesta aplicació client conté tota la lògica i funcionalitat de l'aplicació, per tant, només depèn de si mateixa i d'una connexió amb la base de dades per funcionar en mode autònom, i a més una connexió amb el servei *Terminal Manager* per a interactuar amb el hardware.

Com a mínim s'han de implementar els requisits de IDPresence:

- Operacions amb les dades mestres de IDPresence (operacions *CRUD* (Creació, lectura, actualització i esborrat).
- Gestió d'Incidències
- Gestió de marcatges manuals
- Llistats
- Comunicació amb *Terminal Manager*

Tots aquests requisits són indispensables per poder migrar IDPresence de versió de forma correcta i sense que els usuaris actuals trobin a faltar la funcionalitat.

## 3.2 Anàlisi de l'aplicació futura

### 3.2.1 IDPresence 4.0: objectius

L'aplicació futura ha de corregir els problemes de IDPresence 2.0 i a més proporcionar noves funcionalitats demandades pels clients actuals.

Per tal de corregir els problemes de la versió anterior, la nova aplicació tindrà una estructura de client web – servidor. Aquesta decisió de disseny es va prendre per tal de guanyar en versatilitat i solucionar la major part dels problemes.

L'avantatge principal d'aquest sistema és la facilitat a l'hora de desplegar una actualització:

**Procés actual d'actualització**

1) Parar servei *TM*
2) Parar servei de Base de Dades
3) Actualitzar *SQL* (*)
4) Actualitzar *TM*(*)
5) Actualitzar tots els clients (*)
6) Reiniciar servei *TM*
7) Reiniciar servei *SQL*

**Procés futur d'actualització**

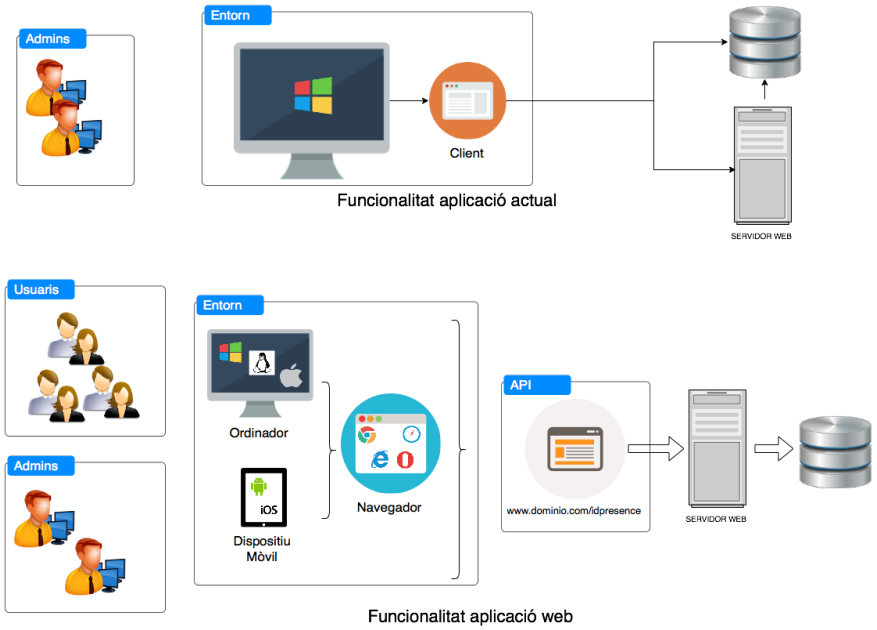
1) Parar servei *TM*
2) Parar servei Web (*ASP*.*net* i *AngularJS*)
3) Parar servei de Base de Dades
4) Actualitzar *SQL* (*)
5) Actualitzar *TM* (*)
6) Actualitzar *Web* (*ASP*.net i *AngularJS*) (*)
7) Reiniciar servei *SQL*
8) Reiniciar servei *TM*
9) Reiniciar servei Web (*ASP*.*net* i *AngularJS*)

**(*)** → Actualitzar si es necessari.

El procés d'actualització actual és un procés de menys de mitja hora pel pas 3 i 4 i una jornada o dues de treball per al pas 5. Això fa que una actualització de *software* sigui molt costosa en temps i diners. Amb el nou procés, els passos del quatre al sis es fan tots tres amb menys de mitja hora.

Un altre avantatge molt important, és la gran quantitat de dispositius que permetran accedir a la informació: des d'un ordinador portàtil o de sobretaula amb *Windows*, *Linux* o *Mac*, passant per un Tablet *Android* o un *iPad* fins a un ordinador tàctil *Surface* amb *Windows*. A més de la diversitat de dispositius, s'intenta donar la major flexibilitat de poder escollir el navegador més còmode per l'usuari (i adaptar-se també a les directrius que marquen algunes empreses a nivell de navegador) i utilitzar *Chrome*, *Firefox* o *Internet Explorer* (amb certes restriccions i utilitzant les ultimes versions, ja que l'esforç de compatibilitzar amb les versions antigues és molt elevat).

A més, amb l'ús de navegadors actuals solucionem un altre problema. Ja no s'ha de guardar informació de connexió amb servidors ni configuracions personals en un arxiu o al registre, ja que totes les configuracions es guardaran a dins d'un emmagatzematge intern que incorporen tots els navegadors actuals i es recolzaran de la Base de Dades, disminuint possibles problemes relacionats amb la interacció d'usuaris inexperts amb els arxius de configuració.




*Il·lustració 7 – Diferències de funcionalitat IDPresence*

La reescriptura del client per adaptar-ho a la nova infraestructura, permetrà solucionar els altres problemes greus que tenia a nivell intern: la compatibilitat amb 64 bits, l'ús de llibreries propietàries i la barreja de funcionalitat al llarg de tota la aplicació.

### 3.2.2 IDPresence 4.0: Nous requeriments

Haver adquirit aquesta infraestructura, ens aporta un medi per desenvolupar un requisit cada cop més demandat al mercat: un portal de l'empleat. Aquesta nova funcionalitat és el requeriment més important que s'implementarà des de zero. Actualment, el consumidor de les dades és un administrador del sistema (figura 7) però això canviarà amb la migració, el consumidor de les dades és un administrador del sistema, un empleat, un cap de departament, etc. Hi haurà diversos rols de consulta i diverses funcionalitats visibles per aquests rols.

Un altre requeriment nou que s'ha d'implementar és la funcionalitat de control d'Accés. L'aplicació actual limita l'accés a 2 estats: accés permès o accés restringit. Per a moltes empreses, aquesta simplificació no es suficient, si no que volen restriccions horàries, de credencials, etc. Per exemple, volen que l'entrada a l'edifici estigui oberta a totes hores, per permetre l'entrada de treballadors a qualsevol hora, però volen que l'accés al menjador només sigui a l'hora de dinar.

# 4 Implementació

## 4.1 Infraestructura: arquitectura client web – servidor

La decisió de triar una estructura client web – servidor és molt clara: versatilitat. Anteriorment, tota la complexitat i còmput de dades es feia al client. Això implicava una sèrie de requeriments mínims de les màquines que havien de executar el client. Per tant, si no separàvem el programa, podíem tenir problemes de rendiment, un augment de la mida dels executables, dificultat a l'hora de les actualitzacions, etc.

Quan separem el client del servidor millorem:

- Rendiment: el servidor és una màquina dedicada d'una potència escalada a les necessitats del client. Això permet que les màquines satèl·lit no cal que siguin molt més potents per a complir amb les necessitats.
- Actualitzacions: si tenim tot el codi als clients, una actualització menor implica el desplegament de tots els clients. Pel contrari, si fem la separació es podria donar el cas de actualitzar només el servidor sense obligar a actualitzar els clients. O viceversa, es podria actualitzar tots els clients sense tocar el servidor.
- Reduïm la mida de l'executable: actualment no és un problema perquè l'emmagatzematge ha reduït moltíssim el seu cost per GB. Aquest tipus de millora actualment no és de gran utilitat ja que l'espai d'emmagatzematge ha augmentat molt als servidors i per tant ha baixat el seu cost, encara que, segueix comportant un avenç que ajuda a la millora de qualitat i de eficiència dels servidors.
- Disminuir costos de manteniment: a l'hora de separar client i servidor, el codi queda separat. Al quedar el codi més organitzat és més fàcil de mantenir. La facilitat de manteniment implica una menor dedicació a l'hora de corregir problemes o realitzar noves funcionalitats, cosa que implica un desenvolupament més ràpid i àgil i això es tradueix en més temps per altres projectes, desenvolupament de més funcionalitats, aprenentatge, etc.

A més, es va decidir que el servidor seria una *API*. Una *API* és un servei web que exposa serveis i dades. Un cop s'han publicats aquests serveis, qualsevol tipus d'aplicació pot consumir dades d'aquests. Actualment, la gran majoria de grans pàgines web les utilitzen per tal de aconseguir una versatilitat molt gran a l'hora de desenvolupar clients natius per a diverses plataformes i separa molt la funcionalitat del client i del servidor.

Un cop decidida la infraestructura física, hem de decidir les tecnologies empleades a les dues parts.

## 4.2 Servidor

### 4.2.1 Framework web: MCV.net


Primer de tot, es va decidir fer servir .net. La decisió d'utilitzar .net és per la tradició a l'empresa. És el llenguatge que s'ha fet servir des de la creació de l'empresa, i això implica que els programadors de l'empresa coneixen el llenguatge, els *IDE*, etc. Això ajuda a tenir una major rapidesa a l'hora d'implementar la nova aplicació i la possibilitat de migrar el codi de l'aplicació antiga de *VB.net* a *C#* (que s'utilitza per la major facilitat per treballar amb el *framework* web escollit) agafant i millorant-lo.

A part d'això, els moviments de Microsoft de portar el *CLR* a una plataforma *Open Source* també han servit per augmentar la confiança amb el .net. Aquest canvi de política de Microsoft ens permetria executar la part de servidor a qualsevol de les plataformes més utilitzades actualment: *Linux*, *OSX* o *Windows*, tot i que se seguirà utilitzant un servidor *IIS* corrent sobre Microsoft *Windows*.

A més de la decisió de utilitzar *.net*, es va decidir la utilització de *MVC.net* i més concretament WebAPI 2. WebAPI es un *framework* per construir *API* construït sobre .net i que permet la construcció de pàgines web. A continuació es detalla el funcionament del *pipeline* (curs d'execució) de *MVC.net*):

1) Nosaltres fem una petició *REST*(*GET*, *POST*, *PUT*, *DELETE*) a un *endpoint* (servei que ens exposa la API que hem creat). Per exemple http://lanostraapi.com/usuaris. Aquesta petició, arriba al servidor web.

2) El servidor web intenta, mitjançant el mòdul *Route*, trobar el servei al codi del qual hem fet la petició. Si el troba, crea un objecte *RouteData* que conté el *RequestContext*, o sigui, crea un objecte que coneix a quin mètode del codi volem anar (*RouteData*) i la informació que hem transmès amb la petició HTTP (*RequestContext*).

3) Un cop coneixem la localització del mètode que volem executar, *MVC.net* s'encarrega de crear una instància del Controlador (classe que conté el mètode que volem executar).

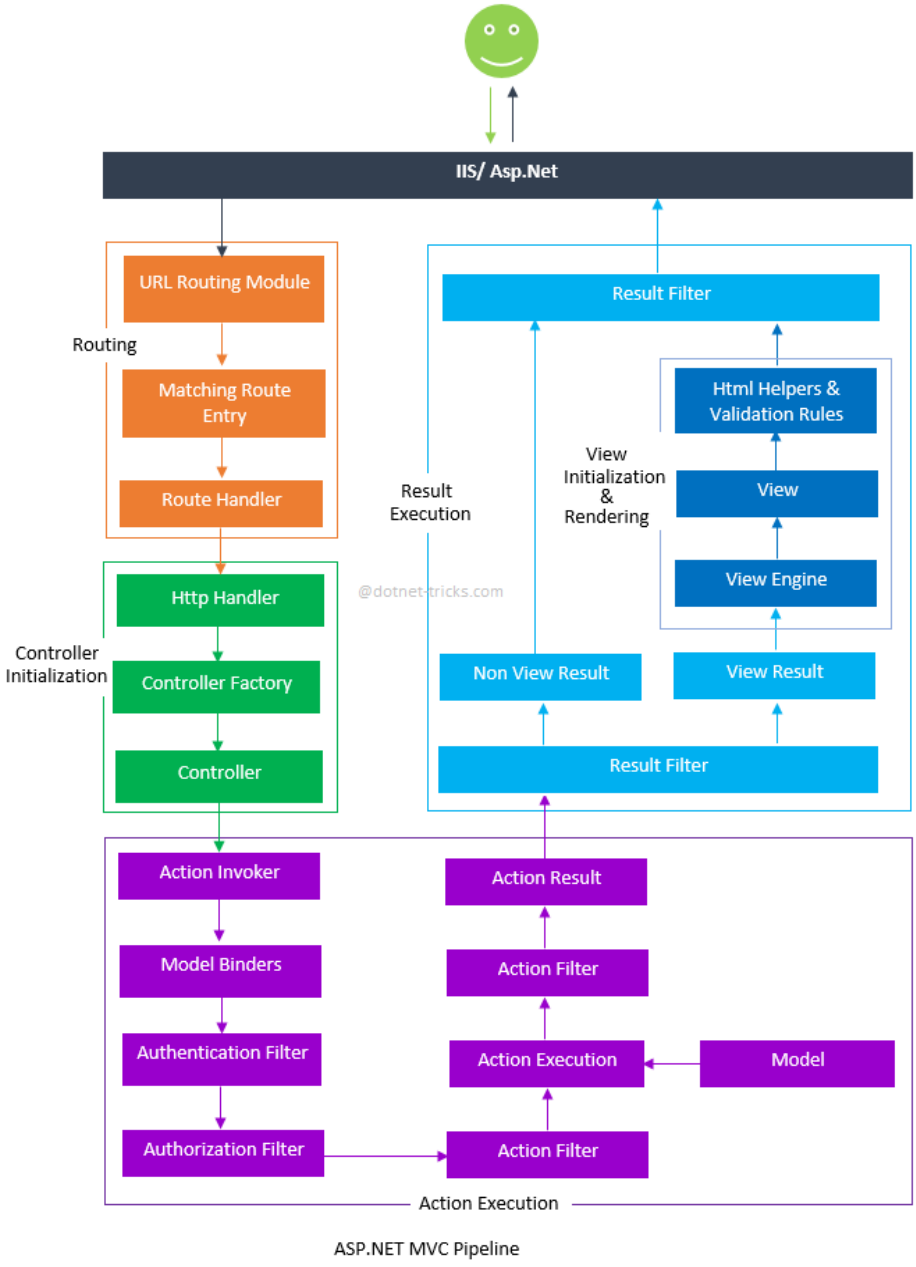
4) Un cop tenim la instància del controlador, autentiquem l'usuari. Autentificar (no confondre amb autoritzar) consisteix en identificar l'usuari que ha fet la petició. Aquesta informació la tenim sempre i quan l'usuari tingui la sessió iniciada al sistema. Si la autentificació és incorrecte, es retorna un codi d'error 401.

5) Si l'usuari s'ha autentificat correctament, comprovem que l'usuari estigui autoritzat per a executar aquest mètode. Per exemple, un usuari normal pot consultar la seva informació, però no la informació d'un altre usuari. Aquest nivell de restricció es pot assignar per usuari, per rol, etc. Si l'usuari no té permisos, es retorna un codi d'error.

6) Si la autentificació és correcte, el següent pas que fa és *bindejar* (copiar) els paràmetres rebuts a la petició *http* a un model (classe que conté tants membres com paràmetres

conté la petició. Aquest pas ens evita tenir que escriure tot el *parseig* de paràmetres (analitzar la informació que rebem del client i extreure tota la que volem) ja que els tenim tots directament en el model.

7) Un cop s'han *bindejat* els paràmetres, s'executa el que haguem programat a dins del mètode que hem cridat (en el nostre cas faríem una consulta a la base de dades ). Quan ja haguem executat el necessari pel mètode, retornem el resultat de les instruccions realitzades en aquest mètode. En el nostre cas, retornaríem una llista dels usuaris de l'aplicació.




*Il·lustració 8 – Pipeline de MVC.net*

Tot aquest procés detallat s'hauria d'haver programat de de si no haguéssim pres la decisió d'utilitzar aquest *framework*. Gràcies a *MVC.net* i *WebAPI* ens hem estalviat la gran part del procés *d'enrutament* de peticions, *bindeig* de paràmetres, autentificació, etc. A més, s'ha seguit la política de *Microsoft* d'obrir el codi font i fer-lo Open Source.

### 4.2.2 ORM de Base de Dades: Entity Framework


Per tal d'accedir a la base de dades, també hem fet la elecció d'un *framework ORM* de Microsoft. Els motius de l'elecció d'un *framework* per accedir a la base de dades és el mateix que el d'haver escollit un *framework* per escoltar les peticions web: estalviar codi, optimitzar el temps i no reinventar la roda. L'escollit en aquest cas ha sigut *Entity Framework 6*. *Entity Framework* és un *framework Open Source* des de la versió 5.0 que ens facilita la obtenció de dades de la base de dades. A part d'aquesta avantatge, *Entity Framework* està integrat a les eines de Visual Studio, és de *Microsoft* i té molt suport a la xarxa.

Utilitzar un ORM té les seves avantatges:

1) Accelerar el desenvolupament eliminant feines repetitives com codificar els paràmetres de les consultes a la Base de Dades, *mapejar* les columnes a objectes, etc.
2) Fer un desenvolupament més abstracte i portable, ja que un *ORM* ja sap com fer consultes a la base de dades sense fer que el programador les escrigui.

Té alguna sèrie de inconvenients com per exemple una reducció del rendiment, però canviant el *ORM* de *ADO.net* utilitzat a l'anterior versió a *Entity Framework* ja guanyem en funcionalitat i rendiment.

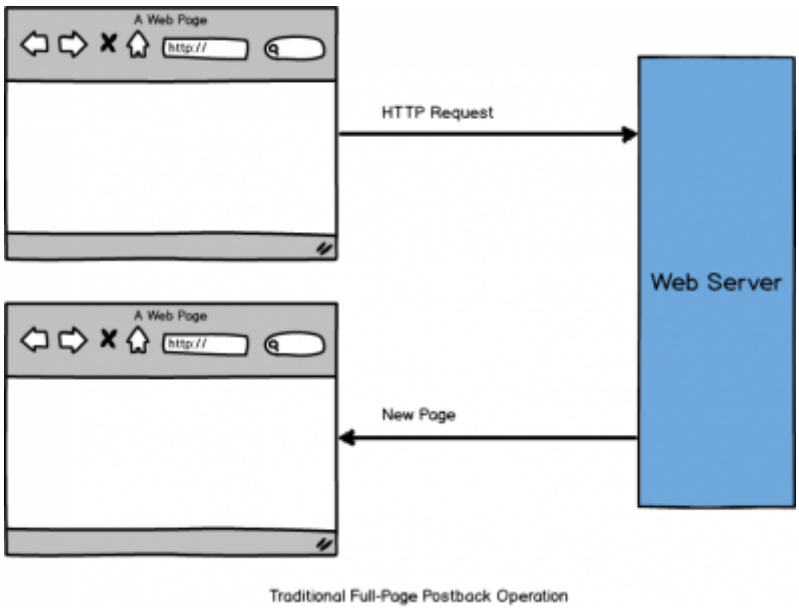
## 4.3 Client


Per la part client hem decidit utilitzar una interfície web com ja hem comentat anteriorment.

Per a programar aquesta interfície web, farem servir varies tecnologies de l'àmbit web com son *HTML*, *JavaScript* i *CSS*. Amb aquestes tres tecnologies, crearem una interfície web atractiva, funcional i amb un bon rendiment que funcioni a qualsevols dels 3 navegadors més utilitzats (basant-nos en varis estudis) actualment com són *Chrome*, *Firefox* i *Internet Explorer* a partir de la versió 9 (ja que les versions anteriors tenen problemes greus de compatibilitat amb les eines que utilitzarem).

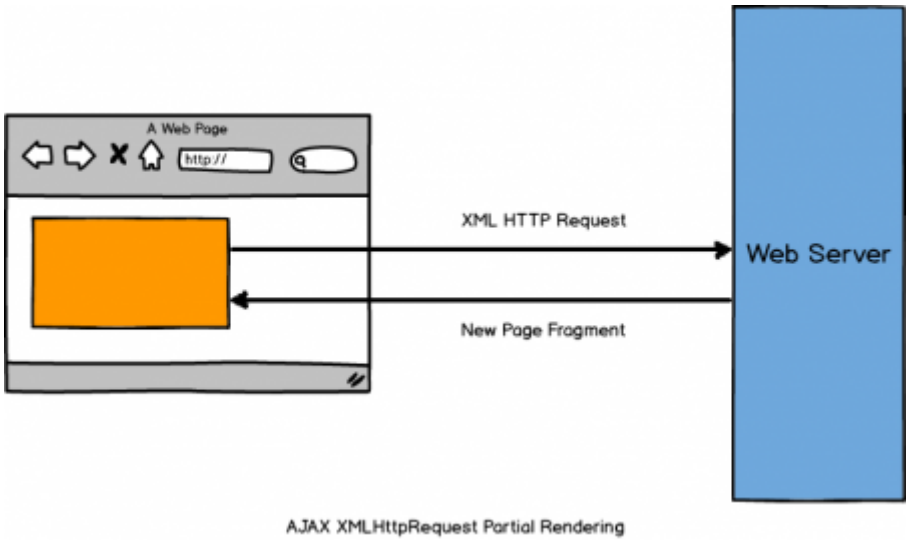
### 4.3.1 AngularJS

Pel client també utilitzarem un *framework* per tal de fer una *SPA*. Històricament, la navegació web recarregava tota la pàgina cada cop que fèiem *click* en un enllaç. Això era perquè eren pàgines estàtiques sense cap contingut dinàmic.




*Il·lustració 9 – Funcionament HTML*

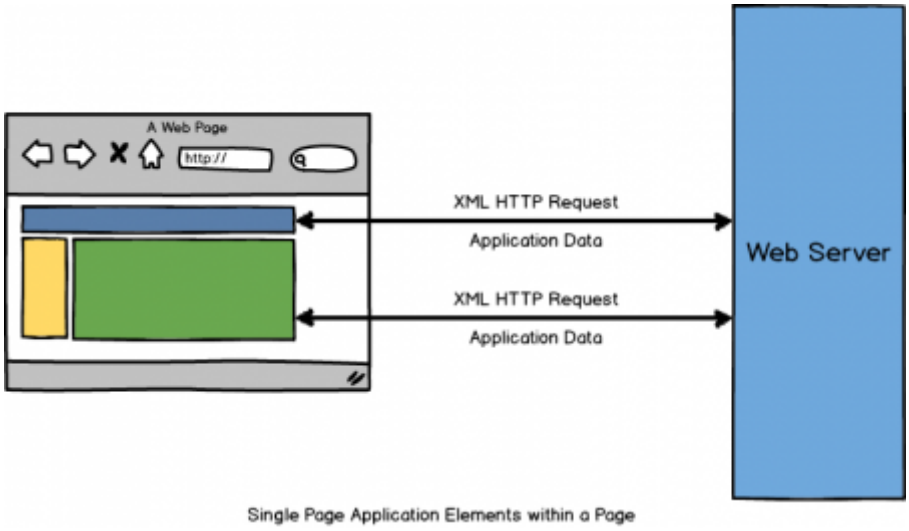
Amb l'aparició de tecnologies com *JavaScript*, això va començar a canviar per fer una web més dinàmica. *Javascript* i eines com *jQuery* permeten modificar la visualització d'una pàgina un cop ja ha estat carregada. Això aporta una flexibilitat major i la possibilitat de programar interfícies més interactives amb l'usuari.




*Il·lustració 10 – Funcionament Web 2.0*

Una *SPA* implica que la web sencera només es carrega un cop (l'inicial). A partir d'aquí, s'utilitza el *DOM* per modificar la composició de la pàgina web. Això té moltíssimes avantatges:

- Sembla una aplicació d'escriptori: al carregar trossos de la pantalla, dóna la sensació de ser una aplicació estàtica.
- Es minimitza el temps de resposta al carregar cada cop només el necessari.
- Es segueix notificant una barra de progrés que indica a l'usuari que està havent-hi una transmissió de dades.
- Es pot accedir des de tot arreu amb una connexió i un dispositiu amb navegador.

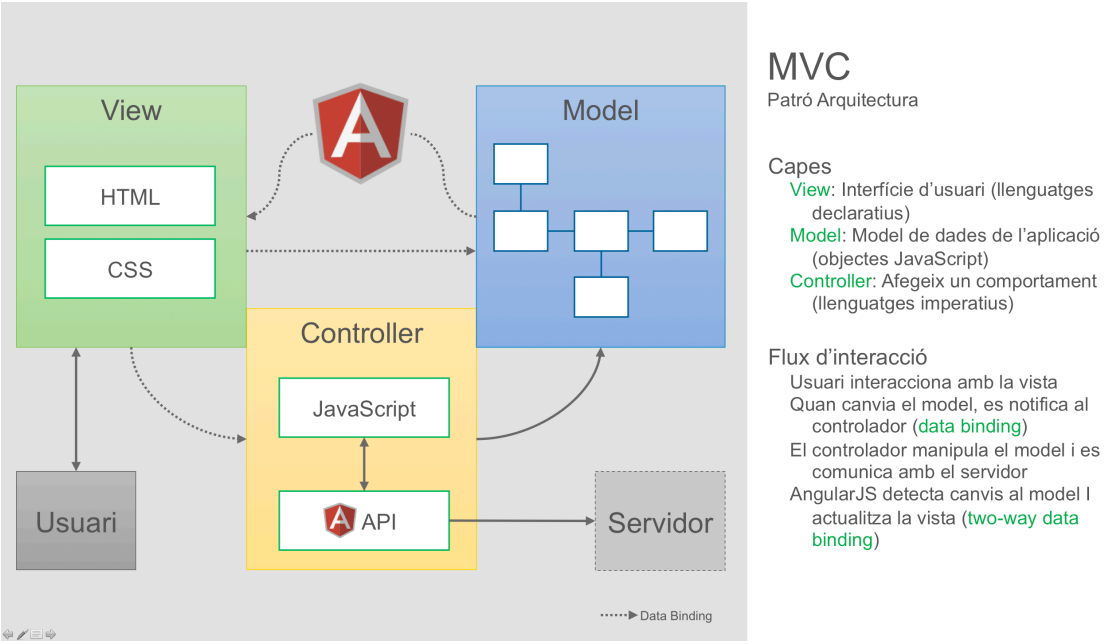



*Il·lustració 11 – Funcionament SPA*

D'entre una sèrie de *frameworks* que faciliten la creació de webs *SPA* he escollit *AngularJS* per varis motius. Entre ells està que és *Open Source*, desenvolupat per *Google* i té el major suport de la comunitat de programadors. A més, *AngularJS* conté totes les eines que necessitem per tal de desenvolupar l'aplicació.

*AngularJS* té moltes avantatges:

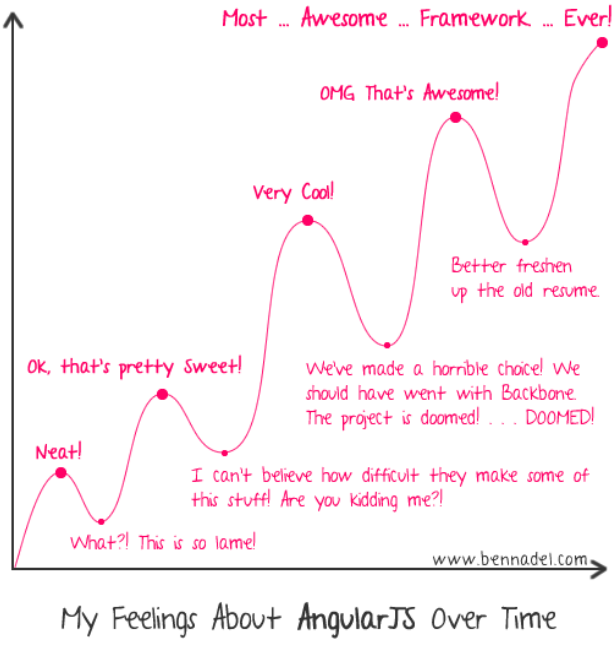
- *Data binding*: permeten *bindejar* un model directament sobre la vista. Això implica que un canvi sobre el model instantàniament es reflectit a la vista i viceversa.
- El suport de la comunitat és molt gran i té un gran numero de plugins Open Source també, com podem veure a http://ngmodules.org/ amb 1410 mòduls.
- Permet separar la lògica de la vista (utilitzant una estructura *MVC*)




*Il·lustració 12 – Patró MVC (Model – View – Controller)*

També té alguns desavantatges:

- La corba de dificultat d'aprenentatge és molt fluctuant
- Dificultat de treballar amb *scopes* (espais de variables diferents per cada controlador).
- Poca documentació




*Il·lustració 13 – Sensacions amb Angular JS*

Tot i els desavantatges, he considerat que *AngularJS* és la millor opció per a la idea que tenia de desenvolupament.

Per tal de facilitar la feina i no reinventar la roda, s'han utilitzat varis plugins *d'AngularJS* com:

- **Angularjs-translate:** era un requeriment important fer que la nostra aplicació fos una aplicació multi-idioma. Per tal de aconseguir-ho, hi havien varies maneres, però es va escollir aquest *plugin* per la integració amb *AngularJS*. AngularJS-translate ens permet la localització de l'aplicació mitjançant fitxers JSON on crearem estructures organitzades amb clau-valor que representen el placeholder i la traducció amb l'idioma que escollim. Gràcies a aquest sistema, només hem de afegir les claus i tindrem una aplicació multi-idioma.
- **Ui-bootstrap:** Bootstrap *CSS* és una fulla d'estils creada pels fundadors de *Twitter*. Aquesta fulla d'estils proporciona facilitat a l'hora de gestionar *layouts*, aplicar controls d'usuari nous, etc. Gràcies a ui-bootstrap, tenim una sèrie de directives que ens estalvien molt de temps d'implementació de controls com *DatePickers*
- **Ui-router:** el sistema d'enrutat de la web bàsic de *AngularJS* és ng-route. Aquest sistema és molt limitat: no permet vistes aniuades, no permet estats, etc. UI-router soluciona totes aquestes mancances afegint un control sobre l'enrutament gràcies a dissenyar la pàgina web com una màquina d'estats. Això permet vistes aniuades, vistes paral·leles, etc.
- **Angular-loading-bar:** per tal de donar a l'usuari un feedback cada cop que es fa una petició al servidor, s'ha escollit utilitzar aquest *plugin*. Gràcies a ell, cada cop que fem una petició, aquesta és interceptada i es genera una barra de progrés sutil que ens proporciona una informació visual del perquè una cosa està trigant.
- **UI-Grid:** un control d'usuari molt comú i molt utilitzat per mostrar informació són els grids. Per tal d'aportar una funcionalitat als *grids* (filtrat d'informació, ordenació per columnes, conteig de files, etc.) s'ha decidit utilitzar *ui-grid*, gràcies a la seva integració amb *AngularJS*, la facilitat de configuració, el disseny amb *plugins*, etc.
- **Ng-storage:** com que estem utilitzant el navegador, tenim certes restriccions d'interacció amb el sistema operatiu per tal de protegir l'usuari de webs malicioses. Per algunes coses, com per exemple guardar la informació de l'usuari, la configuració dels *grids*, idioma, etc. Els navegadors moderns ens proporcionen un enmagatzematge al navegador. Aquest *plugin* ens facilita l'accés a aquest proporcionant un accés clau – valor a dins d'aquest.

# 5 Suport funcional

En aquest apartat descriurem les particularitats d'implementació, juntament amb una explicació de totes les entitats que prenen part a l'aplicació.

## 5.1 Desenvolupament Iteratiu i incremental

Per tal de desenvolupar el projecte, s'ha decidit fer un desenvolupament iteratiu i incremental. Per tal d'aconseguir-ho es van seguir els següents passos: es va fer una planificació inicial del projecte (es van decidir els requeriments funcionals necessaris), i a partir d'aquí va començar el procés iteratiu:

- Definir els requeriments necessaris per aquella iteració.
- Dissenyar i implementar els requeriments necessaris.
- Testejar els requeriments necessaris.
- Avaluar si els requeriments estaven complerts i anotar els problemes de la iteració.

A cada iteració es duien a terme els 4 passos i es feia una valoració de l'estat del projecte. Aquest procés va ser intern durant tota la fase de disseny arquitectònic bàsic. Quan aquesta fase es va acabar, es van començar a provar a clients per tal d'obtenir una font més de *feedback* a part de la del programador. Aquest sistema ha permès tenir una connexió permanent amb el client final per saber la seva opinió i problemàtica i estudiar solucions.

## 5.2 Principis SOLID

Per tal de mantenir un bon disseny de l'aplicació, s'han implementat els principis SOLID:

- **Principi de responsabilitat simple (SRP):** un objecte només té una funcionalitat, o sigui que només s'utilitza per una finalitat. Per aconseguir-ho, s'ha procurat que el codi es dividís en classes amb un propòsit simple, no en classes monolítiques i amb molta funcionalitat. Per exemple, si implementéssim un cotxe no podríem crear un objecte cotxe que contingui tots els elements (motor, rodes, etc.) si no que hauríem de tenir separades totes les parts i tots els comportaments diferenciats.
- **Principi obert – tancat (OCP):** s'ha intentat que la gran majoria de classes puguin ser bloquejades i s'hagin de modificar el mínim de classes a l'hora de fer una correcció. Això s'ha implementat amb una sèrie de classes base abstractes amb un funcionament genèric i aplicable a tots els fills, i una sèrie de classes derivades que implementen els comportaments diferents. A més, moltes classes implementen interfícies amb contractes definits per tal de poder substituir comportaments substituint l'objecte que les executa. Seguint amb l'exemple del cotxe, sabem que el cotxe és un vehicle (la seva classe base). Aquesta classe abstracta, podria contenir el comportament d'un vehicle de forma genèrica, mentre que un cotxe tindria les particularitats. La classe vehicle podria ser bloquejada, mentre que la cotxe seria modificable.

- **Principi de substitució de Liskop (LSP):** s'ha intentat que el màxim de classes derivin d'una interfície que defineixi el comportament de la classe. D'aquesta manera, es podrien canviar classes senceres per derivats seus sense alterar el funcionament. Amb el cotxe, hauria de funcionar d'igual manera amb un seient còmode que amb un seient esportiu, o amb motor de x o de y cavalls.
- **Principi de segregació d'Interfícies (ISP):** s'ha intentat dissenyar una sèrie d'interfícies genèriques que hagin d'implementar les classes per tenir un cert funcionament. Un cotxe hauria d'implementar uns comportaments (acceleració, girar, etc.) o un avió hauria d'implementar uns altres comportaments (guanyar altitud, etc.).
- **Principi d'injecció de dependències (DIP):** com que s'han dissenyat les interfícies per definir comportaments, les classes obtenen els objectes que defineixen comportaments injectant-los. És a dir, els mètodes treballen amb una sèrie d'objectes que implementen uns comportaments, però no són les pròpies classes que els declaren, si no que se'ls hi passa com a paràmetres els objectes amb els que han de treballar. Per exemple, es com si tinguéssim un cotxe sense motor. Sabem que el motor pot realitzar una sèrie de funcions (accelerar, frenar, etc.) però no sabem quin motor tindrà fins que no en diguem el tipus concret. Al moment de crear el cotxe, li posarem un motor qualsevol i el cotxe funcionarà de manera correcte.

Aquest principis simples, fan que el codi sigui fàcilment modificable fent el mínim número de canvis possibles.

## 5.3 Arquitectura 3-Tier

Per desenvolupar aquest projecte hem decidit fer servir una estructura de 3 capes. Aquesta divisió en tres capes ens permet diferenciar comportaments, afegir modularitat i abstracció al projecte i facilitar el manteniment.

Les 3 capes són:

1) Capa dades
2) Capa de negoci
3) Capa d'aplicació

### 5.3.1 Capa de dades

A la capa de dades és on tenim tota la lògica per interactuar amb la base de dades SQL Server 2012. Aquesta capa implementa totes les crides a Base de Dades (consultes, modificacions, eliminacions, etc.). S'han utilitzat tots els mecanismes possibles per tal que canviar la infraestructura de la base de dades sigui el menys intrusiu possible injectant les dependències de la implementació particular SQL Server.

### 5.3.2 Capa de negoci

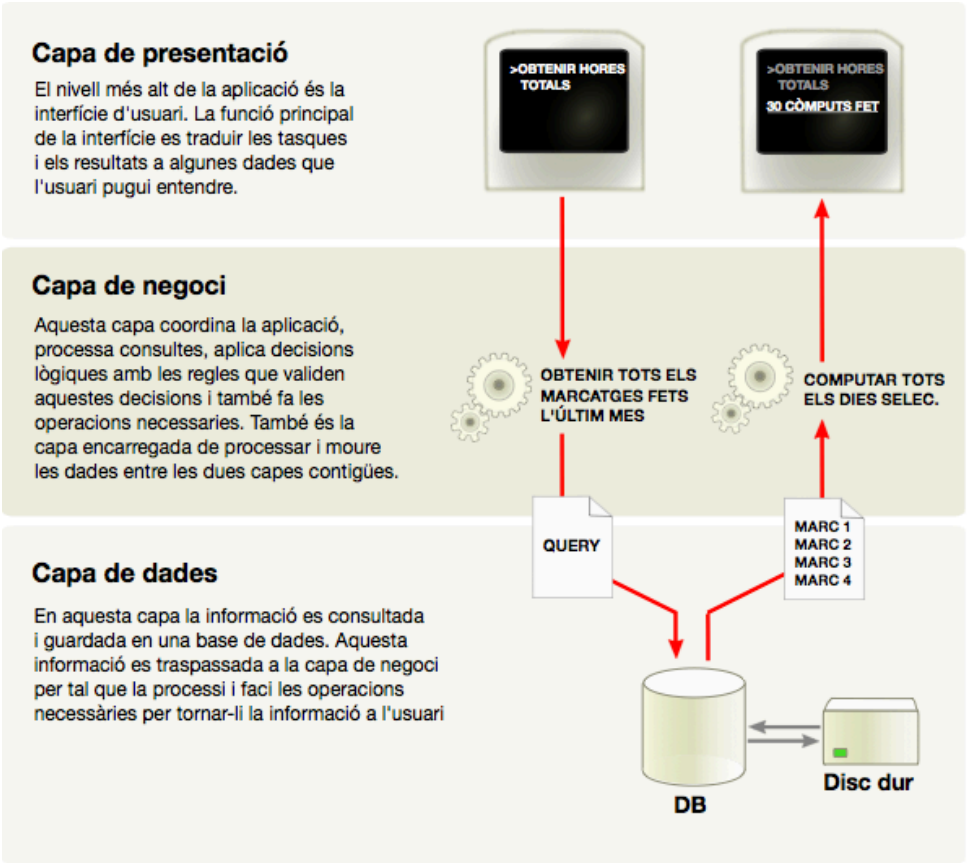
A la capa de negoci és on tenim tota la lògica de la aplicació. Això vol dir que és la capa més important de tota la aplicació, ja que és la que conté totes les regles de negoci de l'aplicació i és la encarregada de recuperar tota la informació necessària per treballar. Aquesta capa és la capa que consulta la informació amb la capa de dades o utilitza altres capes de negoci per obtenir les dades corresponents.

### 5.3.3 Capa d'aplicació

A la capa d'aplicació és on es serveixen tots els mecanismes per tal de treballar amb les dades. És la capa més alta i és on es reben totes les peticions d'usuari de operar amb l'aplicació. Es comunica amb les capes de negoci per tal d'obtenir les informacions o realitzar les operacions que siguin necessàries.

### 5.3.4 Exemples:

Recuperar el compliment d'un usuari:




*Il·lustració 14 – Arquitectura de 3 capes (3-Tier-Layer)*

## 5.4 Entitats de l'aplicació

A més de l'estructura de 3 capes, s'ha fet una quarta capa que conté totes les entitats de l'aplicació. Una entitat és tota aquella representació d'un objecte o concepte del món real que es descriu a l'aplicació. Aquestes entitats tenen totes les propietats com a atributs dins seu, per exemple, el nom, cognoms, etc. d'un usuari són dins de l'entitat Usuari.

### 5.4.1 Usuaris

Un usuari és una persona que pot tenir varis rols dins d'una aplicació. És una persona que té una sèrie d'atributs editables a partir dels quals podrem calcular la seva presència o controlar a quins accessos pot accedir. Pot ser de dos tipus diferents:

- **Empleat**: és una persona que hauria de tenir accés al control de les seves dades mitjançant el portal de l'empleat. A més, aquesta persona tindrà assignades unes credencials per poder identificar-se als terminals, uns permisos per controlar algunes parts de l'aplicació depenent del seu rol a l'empresa (per exemple, un cap de departament pot gestionar les vacances i incidències de les persones al seu càrrec), etc. És una de les entitats més importants i amb més pes de l'aplicació, ja que tindrà pràcticament totes les altres entitats de l'aplicació associades.
- **Visitant**: és una persona que pot o no tenir un accés permanent a l'aplicació (i per tant unes credencials assignades permanentment o alguna cosa més temporal com dispositius RFID) i que no ha de tenir associat tantes entitats, per tant tindrà perfils més restringits. De les visites majoritàriament controlarem la quantitat d'accessos permesos. Quan les credencials no són permanents, s'assignen a l'usuari d'un conjunt de dispositius RFID permanents.

### 5.4.2 Credencials

Les credencials d'un usuari son totes les maneres d'identificar a una persona. A IDPresence, es poden configurar fins a 6 tipus de credencials:

- **RFID:** es pot configurar un únic codi RFID de una de les 3 tecnologies suportades pel Terminal Manager: 125Mhz, Mifare o Mifare Desfire. La lectura d'aquest codi es fa mitjançant uns lectors de cada tecnologia connectats a l'ordinador. Molts lectors funcionen amb COM (difícil de gestionar via web) o com a emulació teclat (que retornen un codi depenent de la configuració interna que posem al dispositiu).
- **Password:** podem configurar un password per tal d'identificar-nos als terminals.
- **Codi de barres:** podem configurar un identificador que generi un codi de barres que podem imprimir en una targeta de tal manera que un terminal ho pugui llegir.

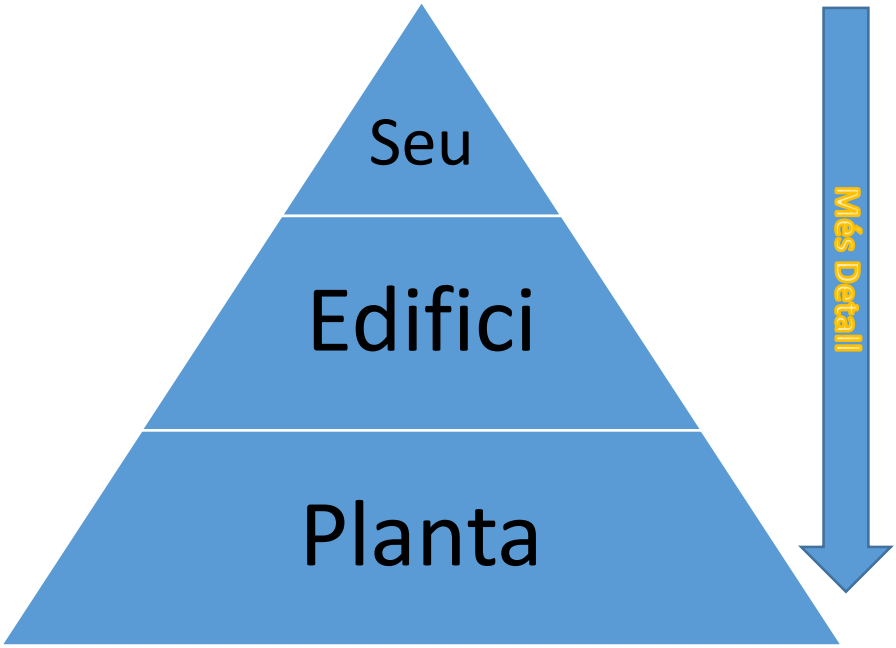
- **Banda magnètica:** igual que amb el RFID, podem assignar un codi llegit d'un lector COM o emulació teclat
- **Empremta:** podem utilitzar un lector d'empremtes per tal d'agafar les característiques de les empremtes. Amb la aplicació web és difícil d'obtenir les empremtes, ja que s'agafen a través de les SDK de cada fabricant i s'utilitzen llibreries dinàmiques dll. Les credencials biomètriques son les úniques que no es poden obtenir des de la web i s'ha d'utilitzar un petit programa per guardar-les a base de dades.
- **Arbre de venes:** podem utilitzar un lector d'arbre de venes per agafar les venes de la mà. Teòricament és el mètode més segur, però també el més costós.

### 5.4.3 Seus, Edificis i Plantes

Una Seu és el nivell més alt d'agrupació geogràfica de persones. Es un concepte mantingut des dels principi de l'aplicació i modela la unió de diferents entorns geogràfics.

Un edifici és un subconjunt de seu i modela un edifici dins d'aquesta seu. Aquest concepte delimita encara més la posició d'una persona.

El nivell més alt de detall i més baix d'agrupació es una planta. Modela la planta d'un edifici i dóna el màxim de precisió donat per l'aplicació.




*Il·lustració 15 – Esquema Seu – Edifici – Planta*

Són uns conceptes que ajuden a delimitar recintes per tal de permetre llistats de localització de persones gràcies a la delimitació dels accessos. Imaginem un recinte gran com el Parc de la Ciutadella de Barcelona. Aquesta seria la seu. Dins del parc hi ha varis edificis com el Parlament i l'Institut Verdaguer. I aquests edificis tenen plantes. Doncs si poséssim un terminal a cada un

dels accessos podríem arribar a saber la ubicació d'una persona de manera força precisa i ràpida gràcies als marcatges que ens enviarien aquests terminals i la ubicació física dels mateixos.

Aquest sistema ja funciona amb molta precisió a una empresa farmacèutica de Barcelona que controla accessos i presència amb el programa.

### 5.4.4 Departaments

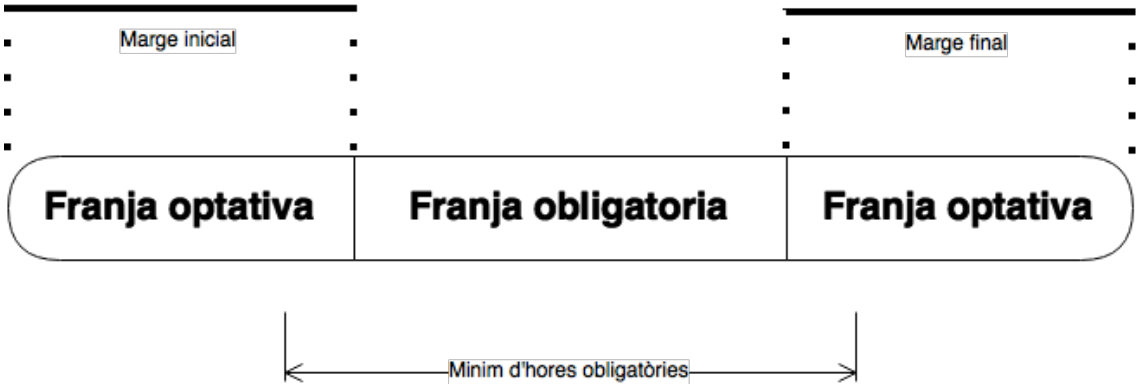
Un departament és una agrupació de persones a nivell humà i que formen un grup de treball. Aquests departaments depenen d'una o varies seus i comparteixen informació (per exemple calendaris de presència). és una agrupació important a nivell de gestió (ens permet donar privilegis de gestió de vacances, d'incidències, etc.) i a nivell de llistats (per donar per exemple compliments per departament, llistats de puntualitat, etc.).

### 5.4.5 Calendaris, tipus de dia i franges horàries

Els calendaris son la eina que ens permet planificar la jornada laboral teòrica dels treballadors a l'empresa. Un calendari no és més que un tipus de dia assignat a un dia concret.

Un tipus de dia és una entitat que ens modela una jornada laboral, per a poder definir el comportament esperat d'un usuari. S'identifica per un nom i una descripció, i està format per una sèrie de franges horàries que formen la jornada prevista que hauria de realitzar aquest dia.

Una franja horària és un interval de temps dissenyat de la següent manera:




*Il·lustració 16 – Esquema franges horaries*

Aquesta definició d'una franja ens permet modelar diferents casos de franges horàries:

- Una franja on el marge inicial i el marge inicial és 0, és una **franja obligatòria,** per tant l'usuari ha de fer les hores exactes per a complir.
- Una franja on hi ha un marge inicial / final / ambdós permet a un usuari fer **hores extraordinàries** dins d'aquestes franges optatives i a més, les hores obligatòries.

- Una franja sense marge inicial / final / ambdós però amb un mínim d'hores permet fer una **jornada flexible** de duració mínima igual a la especificada
- Una franja amb un marge inicial / final /ambdós i a més un mínim d'hores obligatòries permet fixar un horari de presència obligatòria i a més una obligació de presència flexible a escollir per l'usuari, dit amb altres paraules, l'usuari haurà de complir el mínim d'hores establert, que haurà d'estar dins de la franja obligatòria, però la diferencia entre el mínim establert i la duració de la franja la podrà realitzar a conveniència.

Aquests calendaris poden ser per seu, departament o usuari. Aquesta jerarquia s'ha definit per permetre un mecanisme bàsic de gestió de calendaris a nivell grupal.

### 5.4.6 Registres

Els registres son la base de l'aplicació. Un registre és una marca horària i física d'una presencia, es a dir, la prova que retorna el Hardware quan un usuari utilitza la seva credencial per accedir i/o fitxar en algun lloc concret.

Gràcies a aquestes dades, podem conèixer tota la informació necessària per tal de saber el compliment o absentisme d'un usuari, la ocupació actual d'un recinte, etc.

Aquestes dades són guardades pel servei Terminal Manager quan un terminal els hi envia. Gràcies a la configuració dels terminals o una lògica aplicada pel servei, podem diferenciar entre marcatges d'entrada, sortida o accés i a més utilitzar-los o no per a fer tots els càlculs de presencia, compliment, etc.

# 6 Beneficis de la migració tecnològica

Aquesta migració tecnològica ha tingut varis beneficis a tots els nivells:

1) Facilitat d'instal·lacions i Update de l'aplicació: els canvis en infraestructura han fet que sigui una aplicació més senzilla d'actualitzar i d'instal·lar.
2) Atractiu visual: gràcies als canvis realitzats, s'ha millorat la interfície d'usuari i la usabilitat de la aplicació. Això ha fet que els usuaris actuals estiguin contents amb el canvi i els nous vegin un nou reclam per aquest *software*.
3) Abaratiment de costos: gràcies a l'eliminació de llibreries propietàries, s'ha aconseguit una baixada de costos que implica una millor competitivitat.
4) Facilitat de manteniment: la migració ha fet que el manteniment i ampliació del software sigui un procés més senzill i ràpid que amb l'antiga estructura.
5) Independència de l'entorn d'execució: al ser una aplicació web, eliminem totes les restriccions a nivell de sistema operatiu. Per tant, ens es igual Windows, Linux o Mac i si son de 32 o de 64 bits, cosa que abans condicionava la majoria d'instal·lacions.

Tots aquests beneficis compleixen amb l'objectiu del projecte.

# 7 Test de qualitat

## 7.1 Tests privats

Durant el desenvolupament de l'aplicació s'han anat fent tests individuals de funcionalitat i funcionament. Aquests tests faran que els errors més greus de funcionament es solucionin abans de la posada en marxa del sistema.

A més, s'han dissenyat una sèrie de tests de funcionament bàsic per cada pantalla:

- Alta de nou registre
- Modificació de registre
- Baixa de registre
- Registres incorrectes

Tots aquests tests, els he passat jo i després altres programadors de l'empresa. A mesura que s'anaven trobant *bugs*, s'han anat solucionant per tal de fer una primera proba pilot amb un client final i real, i veure el funcionament en un entorn real.

## 7.2 Tests a usuaris nous

Per a la segona fase del test, s'ha escollit un client nou (amb un altre tipus de sistema de control d'accés i presència) al qual s'ha fet un estudi de viabilitat del projecte per veure si el programa compleix les seves necessitats.

Un cop s'ha constatat que el programa compleix les expectatives i funcionalitats necessàries se'ls ha proposat un contracte pilot per testejar l'aplicació.

Aquest pilot s'ha basat en fases:

1. Fase de implantació: se'ls instal·la el servidor (web i API) perquè puguin començar a treballar amb el sistema. Per tal de fer un paral·lel amb l'aplicació anterior, no es desplega el Terminal Manager, si no que se'ls hi han migrat les dades a la nostra Base de Dades.
2. Fase de transició: un cop ja es tenen totes les dades introduïdes al nou sistema, es procedeix a desplegar el Terminal Manager. Un a un, es van desconnectant els terminals del sistema anterior i es van afegint al nou sistema, traspassant les polítiques horàries, d'accés i credencials dels usuaris als terminals. Aquest procés es duu a terme per tal d'assegurar una transició suau. Com que els terminals controlen relés que obren portes, un pas en fals pot portar a un bloqueig general de tots els accessos o un funcionament erràtic dels terminals, cosa que portaria a una desconfiança amb el nou sistema.
3. Fase de convivència: durant un temps, conviuran els dos sistemes en paral·lel. Això vol dir que les dades s'agafaran amb el Terminal Manager i amb el seu programa.
4. Fase final: un cop s'han passat tots els terminals al nostre sistema i ja no són necessàries les dades del programa antic, es procedirà a desactivar els sistemes antics.

5. Fase d'acompanyament: un cop el client estigui satisfet amb la migració, es produeix un període d'acompanyament de duració variable, depenent de l'acceptació del programa. Aquest acompanyament és a nivell d'atenció activa per part de l'empresa, comprovant que el sistema funcioni de manera correcte i gestionant les possibles faltes d'enteniment del programa solucionades amb formació o petites falles que s'afegeixen a la llista de errors a corregir.

Aquest procés de test s'ha dut a terme un parell de nous clients (dues empreses del Vallès; una de productes químics i una altre de productes congelats) amb bones sensacions. Hi ha una tercera empresa en proves i que és un procés de migració complet de IDPresence 2.0 a IDPresence 4.0 que està a l'última fase del procés.

# 8 Conclusions

D'aquest projecte puc extreure una gran varietat de conclusions gràcies a l'aplicació en un projecte pràctic al qual treballo en la meva etapa professional actual. Per altra banda, és interessant les moltes visions i aspectes en els quals seguir millorant per investigacions futures en aquest entorn.

Primer de tot faré uns comentaris dels aspectes que m'han servit d'una manera més personal a l'hora de treballar en aquest projecte:

Per una banda, aprendre a gestionar un projecte i tot el que comporta (reunions de desenvolupadors i clients, anàlisi de requeriments i funcionalitats, etc.) Poder fer un anàlisi exhaustiu de les tecnologies que s'utilitzaran pel desenvolupament del projecte perquè siguin suficients per al desenvolupament actual i futur de l'aplicació. Fer un anàlisi de funcionalitat igualment d'exhaustiu, ja que el nou sistema ha de fer com a mínim, tot el que feia el sistema antic perquè els clients actuals no trobin mancances i ha de tenir funcionalitats afegides per tal de justificar la inversió passada i futura en l'aplicació.

Pel que fa a l'estructura del *software*, he pogut analitzar i aplicar el màxim de patrons de disseny de *software*, fent-lo el màxim de modular, aplicant tots els principis SOLID per tal de poder modificar el sistema de forma senzilla i d'afavorir el manteniment del *software*.

Seguidament, les reunions amb clients per tal de saber que volen a la fase de anàlisi i per saber les opinions a la fase de test i integració. Saber gestionar de forma correcte els requeriments concrets que demanen certs clients per tal de no caure en un "pou sense fons" de requeriments que augmentaran el temps de desenvolupament.

A més, a nivell de la migració del *software*, es pot concloure que el procés descrit per desenvolupar la migració del software és correcte i es pot aplicar a qualsevol projecte informàtic i finalitzar amb un resultat correcte. Actualment, la migració està funcionant a varis clients nous (tenien dues plataformes diferents i s'han migrat a IDPresence) i un client al qual s'ha fet una migració per problemes amb els nous sistemes operatius que venen amb els ordinadors nous. Aquestes implantacions estan donant bons resultats i funcionen de forma satisfactòria.

Als usuaris actualitzats els ha agradat no tenir una aplicació més d'escriptori i les noves funcionalitats de l'aplicació, a més de la facilitat de rebre les actualitzacions que s'han anat realitzant periòdicament per corregir *bugs* de funcionament i mancances de funcionalitat.

Al usuaris migrats d'altres plataformes els ha agradat la interfície senzilla i web, i les solucions preses per proporcionar les funcionalitats demanades.

Als treballadors se'ls està donant accés de forma esglaonada per testejar la funcionalitat del portal de l'empleat, ja que al ser la part més nova s'ha decidit fer una implantació lenta i segura.

A nivell de negoci, la migració tecnològica està essent un nou reclam per als clients, ja que moltíssima gent treballa a nivell empresarial amb tabletes per la facilitat de mobilitat i el descens del preu envers els ordinadors portàtil clàssics, i per tant, encaixa amb la migració que s'ha fet a l'aplicació. Gràcies a això, s'ha donat una empenta a aquest software que ja semblava antiquat i mancat de funcionalitat. Per tant, l'aplicació de la migració ha sigut molt positiva per l'empresa i pot ser-ho més en un futur proper.

A continuació es resumeixen les conclusions que s'han extret del desenvolupament de la actualització.

| Sistema antic | Solució |
|---|---|
| Impossible actualitzar en Sistemes Operatius de 64 bits o en sistemes que no siguin Windows | Utilitzar una solució web ens permet l'execució independentment del tipus de Sistema Operatiu. |
| Llibreries propietàries | Utilitzar solucions *Open Source* que ens permeten eliminar despeses en aquestes llibreries |
| Llibreries obsoletes | Utilitzar la versió actualitzada de les llibreries o substituir-la per versions més noves ens permet solventar els problemes |
| Winforms (dificultat a l'hora de treballar en paral·lel) i Devexpress | Utilitzar una solució web sense generació de codi ens permet una manera més senzilla de treballar de forma paral·lela |
| Infraestructura clients – base de dades – Terminal Manager | Utilitzar una infraestructura client – servidor on els clients són una web i el servidor connecta amb base de dades i Terminal Manager ens permet facilitar actualitzacions i gestionar configuracions |
| Dificultat d'actualització | La decisió de fer un sistema web ens permet facilitar les actualitzacions ja que les centralitza en 2 punts (servidor i base de dades) |
| Manteniment de l'aplicació | L'aplicació dels principis SOLID, la distribució en capes, el patró MVC, etc. fan que l'aplicació sigui més fàcil de mantenir i de millorar ampliant funcionalitat |
| Interfície visual | S'han utilitzat una sèrie de comportaments i components visuals que fan que l'aplicació llueixi i funcioni d'una forma més moderna i més amigable amb l'usuari |

El projecte ha tingut varies complicacions durant la seva execució: s'ha hagut de reescriure tota la aplicació i això es un procés molt complex que ha fet que la fase de disseny prèvia hagi sigut molt dura i amb varis girs i reestructuració de codi. A més, la part visual (AngularJS) ha sigut força complicada per aconseguir la modularitat desitjada i perquè es un framework amb una corba d'aprenentatge força irregular. Aquestes coses han fet que el desenvolupament fos una mica més lent del que es pensava en un principi, però han fet que es tingui una solució més robusta.

# 9 Evolució futura

Gràcies al sistema escollit, el sistema pot tenir un evolució futura molt gran.

Per exemple, gràcies al haver escollit una API de programa, es permetria fer aplicacions natives per a qualsevol sistema operatiu. Hem comentat anteriorment que és una desavantatge tenir una aplicació lligada a un sistema operatiu, per tal de realitzar actualitzacions i tot, però un sistema natiu permetria, per exemple, l'ús dels sistemes biomètrics amb el quals s'estan dotant molts telèfons o tabletes o ordinadors d'escriptori. Gràcies a que es una arquitectura *REST* es permetria implementar només funcionalitats puntals, com marcatges online amb biometria, marcatges geolocalitzats, ús de *NFC*, etc. El més problemàtic és que el navegador restringeix tot l'accés al hardware cada cop més i més per evitar filtracions de seguretat. L'ús de *REST* permet separar molt la funcionalitat i fa que es simplifiqui molt la creació de aplicacions senzilles de consum i modificació de dades.


Un altre exemple seria la implementació d'alertes des del servidor. Aquestes alertes serien per exemple de baixes de persones, notificacions de vacances, notificacions d'incidències, etc.

La modularitat del sistema i la arquitectura client – servidor amb capes, permetria també escalar el sistema en varis servidors: un servidor amb el *Terminal Manager*, un servidor amb el client web, un altre servidor amb el SQL, un altre amb la *API*, un altre d'autenticació... Això permet escalar el sistema, cosa que amb l'aplicació actual només es podria dividir en terminal manager i servidor SQL.

Una evolució futura molt important i que actualment està en boca de tothom es la possibilitat de treballar al núvol (*Cloud*). Al ser una aplicació web, es pot fer que el servidor sigui vist des de l'exterior per tal de tenir una manera d'accedir a les dades de manera global.

# 10 Annex

## 10.1 Glossari

**UI (User Interface):** la UI es la interfície de usuari, es a dir, tots els aspectes visuals d'una aplicació (disseny de pantalles, tipografia, maquetació, etc.).

**UX (User eXperience):** la experiència d'usuari es la manera en que l'usuari interacciona amb la UI, la usabilitat. Una aplicació pot ser bonica i poc intuïtiva i usable (bona *UI*, mala *UX*) o poc atractiva visualment però funcionalment molt rica i molt senzilla (mala *UI*, bona *UX*).

**Winforms:** *Windows Forms* (o formularis *Windows*) és el nom donat a la interfície de programació de Aplicació Gràfica (*API*) inclosa com a part de .net *framework* de Microsoft. Ens proporciona accés als elements natius de la interfície de *Windows* fent un envolcall sobre la *API* de *Windows*, facilitant-nos l'accés a nivell natiu al Sistema Operatiu.

**Control d'accés:** consisteix en controlar mitjançant personal, terminals, barreres, torns, etc. l'accés a determinats recintes utilitzant les credencials de l'usuari. Aquestes credencials poden ser de qualsevol tipus: *RFID*, biomètriques (empremta, vascular), contrasenyes, etc.

El control d'accés ens permet controlar la entrada i sortida de personal autoritzat a diferents zones d'un recinte en temps real. El sistema es basa en una xarxa de terminals de control connectats entre si a una base de dades gestionades per un servidor. El sistema pot funciona en xarxes local o en *VPN's* que simulin xarxes locals. El sistema identifica l'usuari, li dóna o denega accés comparant amb les regles establertes i si està online envia el marcatge al servidor. Un cop guardats al servidor, aquests marcatges es poden consultar per a operar amb ells. En cas de no estar online, els marcatges es desen als terminals físics i s'envien tant bon punt es restableix la connexió amb el servidor.


**Control de presència:** consisteix en controlar mitjançant terminals, persones, etc. si una persona ha accedit a un recinte. Aquest control de presència permet identificar el nombre d'usuaris d'un recinte, calcular les hores productives o absències d'un usuari, etc.

El control de presència ens permet controlar les actituds horàries dels empleats. Es basa en les dades recol·lectades pel control d'accés, per tant podríem dir que requerim d'un control d'accés per tal de realitzar un control de presència.

**Terminal**: Els terminals són equips electrònics que gestionen de manera automàtica l'apertura segura de sistemes d'accés: portes, barreres, torns, valles de pàrquings, etc.

Existeix una àmplia gama de dispositius de control d'accés amb múltiples característiques diferenciadores, però normalment estan compostos per un hardware de control i una interfície d'usuari que pot ser des d'un simple LED per mostrar estats (esperant, pas concedit, restricció de pas, etc.) fins a un display LCD que mostri la biometria recollida per a realitzar a comparació, passant per teclats o targetes que identifiquin l'usuari per a realitzar identificacions 1:1. Aquests terminals tenen unes software intern al qual es realitza la interacció mitjançant una SDK proporcionada pel fabricant del software.

**Terminal On-Line:** terminal que depèn d'una connexió amb un servidor per tal de treballar.

**Terminal Off-Line:** terminal que pot funcionar de forma autònoma un cop se li han proporcionat totes les informacions per funcionar (credencials, permisos d'accés, etc.).

**IDE**: un IDE (*Integrated Development Environment* o entorn de desenvolupament integrat) és un editor de text amb funcionalitats extres afegides (auto completat de text, trossos de codi, errors de sintaxi, etc.).

**Framework**: es un entorn de desenvolupament complet. Conté totes les eines necessàries per treballar amb ell (llibreries, compilador si fos necessari, etc.).

**SDK**: una SDK (*Software Development Kit* o Kit de Desenvolupament del Software) és un conjunt d'eines de software que ens permeten la creació d'aplicacions que interactuïn amb un software o un hardware del que desconeixem el funcionament intern. Aquestes *SDK* inclouen demos, biblioteques, compiladors en cas que fossin necessari, documentació, etc. Aquestes *SDK* contenen el més important que es l'accés a la *API*.

**API**: una API (*Application Programming Interface* o Interfície de Programació d'Aplicacions) és un conjunt de declaracions de funcions i procediments que permeten interactuar amb una plataforma determinada, ocultant com si fossin caixes negres tota la implementació interna de la funcionalitat. Un programador només pot proporcionar i/o demanar informació a la *API* i rebrà com a resposta el resultat de la operació i/o les dades que hagi demanat.

**Credencial:** una credencial es una manera d'identificar un usuari. Pot esser qualsevol tipus: un document identificatiu amb fotografia, una targeta *RFID*, una marca biomètrica, etc.

**RFID**: el *RFID* o (*Radio Frequency IDentification* o identificació per radiofreqüència) és un sistema d'emmagatzemament i recuperació de dades remotes a etiquetes, targetes *RFID*. El principal propòsit de la tecnologia es transmetre la identitat d'un objecte a través d'ones de radio.


**Biometria d'empremta:** la tecnologia biomètrica d'empremta és la més estesa a l'actualitat per la seva maduresa, el seu baix cost, la utilitat i la rapidesa de identificació. Hi ha varies tecnologies per tal de realitzar l'obtenció de la biometria: poden ser òptiques (escaneig de l'empremta utilitzant la llum i el contrast entre les crestes papil·lars i les valls creades entre crestes), ultrasons (utilitzen ones sonores que penetren la capa epidèrmica) i capacitatives (utilitzen sensors transductors que detecten una diferencia de tensió entre les crestes i les valls i construeixen una reconstrucció amb aquestes diferencies de tensió). Tots els sistemes generen un patró (*template*) que conté les característiques més significatives de l'empremta utilitzant algoritmes. La biometria d'empremta és un dels sistemes més utilitzats quan la seguretat, rapidesa i comoditat son qualitats necessàries, ja que es un sistema molt precís (el nombre de falsos positius i negatius es molt baix) i assegura una identificació personal que no s'obté amb altres sistemes ni es pot suplantar com targetes o codis d'accés. Segons el tipus de tecnologies d'obtenció té alguns problemes amb persones amb les empremtes molt desgastades, durícies o envelliment de la pell, però qualitat preu es la millor opció per una identificació segura.

**Biometria Vascular:** la biometria vascular es basa en utilitzar l'esquema de venes del dit com a patró biomètric per identificar una persona. El patró vascular es una característica interna que no varia amb l'edat i que no té afectacions de desgast o dany del dit, però es més lenta i cara que la biometria d'empremta.

**Identificació 1:N:** el procés d'**IDENTIFICACIÓ** és un procés de combinació 1-a-molts (1:N), o sigui l'usuari no precisa confirmar qui és per a realitzar l'accés. El procés obté una nova mostra de lla identificació i les compara amb les ja existents a la base de dades de credencials dels usuaris. Quan es troba una combinació, l'usuari es identificat com un usuari preexistent, o sigui, el sistema localitza una credencial que coincideix amb alguna persona.

**Verificació 1:1:** El procés de **VERIFICACIÓ** es un procés de **combinació** de **un-a-un** (1:1). L'usuari confirma qui és. El procés consisteix amb la introducció d'un identificador d'usuari al sistema i proporcionar una credencial al sistema. Un cop es verifica que l'usuari introduït té assignada la credencial introduïda, es produeix la verificació d'identitat al sistema i se li concedeix l'accés.

**Marcatge / fitxatge:** cada un dels registres guardats al terminal quan un usuari es verifica o identifica. Relaciona un usuari amb un terminal i una hora d'accés pel mateix.

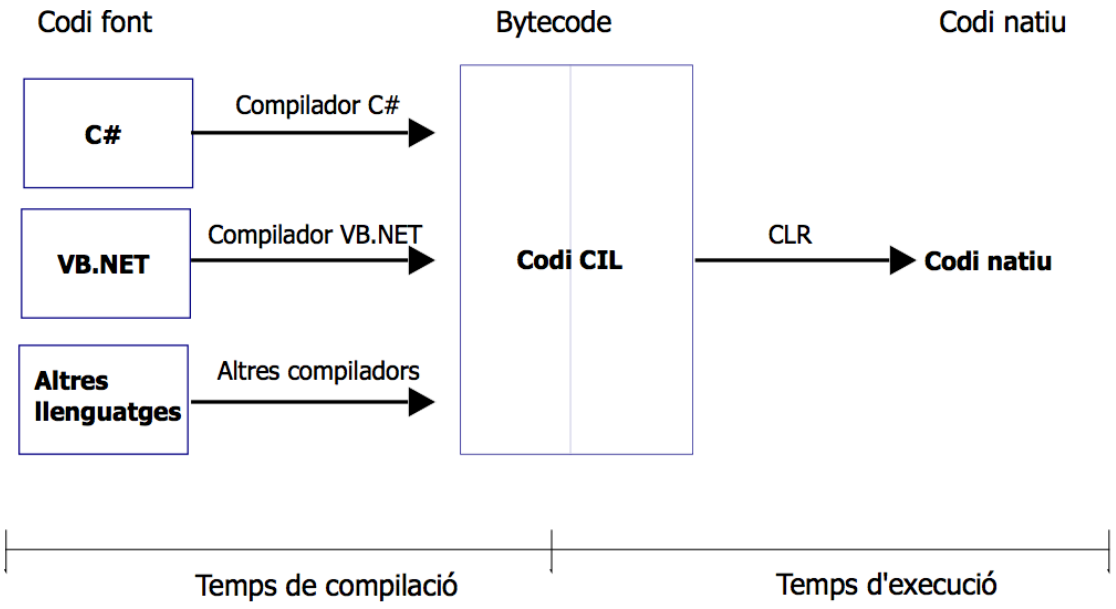
**ADO.NET**: conjunt de classes que exposen serveis d'accés a dades per a programadors. Està inclòs al framework de .net i s'utilitza per facilitar l'accés a la base de dades.

**Plugin**: són petites aplicacions que complementen l'aplicació principal amb funcionalitats noves i molt específiques. Permeten que es pugui fer una aplicació modular amb un petit nucli que conté la funcionalitat bàsica i mòduls (*plugins*) que augmentin la funcionalitat.

**WCF**: WCF (*Windows Communication Foundation)* permet una comunicació entre aplicacions .net de forma ràpida i senzilla pel programador. Permet intercanviar objectes de forma "transparent" entre aplicacions i també llençar esdeveniments de forma remota mitjançant connexions punt a punt.

**CLR**: el CLR (*Common Language Runtime* o Entorn en temps d'execució de llenguatge comú) és la màquina virtual encarregada de traduir el CIL a codi natiu.

**CIL**: el CIL (*Common Intermediate Language* o llenguatge comú intermedi) és el resultat de la compilació de programes .net.




*Il·lustració 17 – Explicació compilació .net*

**Open Source**: *l'Open Source* o Codi obert es tot aquell software o hardware distribuït i desenvolupat lliurement. Això vol dir que es focalitza ens els beneficis de qualitat envers els beneficis econòmics.

## 10.2 Webografia


[1] https://msdn.microsoft.com/en-us/library/dd381612%28v=vs.100%29.aspx

[2] http://en.wikipedia.org/wiki/Representational_state_transfer

[3] http://blog.4psa.com/an-intro-into-single-page-applications-spa/

[4] http://karwin.blogspot.com.es/2009/01/why-should-you-use-orm.html

[5] http://www.dotnet-tricks.com/Tutorial/mvc/LYHK270114-Detailed-ASP.NET-MVC-Pipeline.html

[6] http://www.circuitstoday.com/working-of-fingerprint-scanner-2

[7] http://www.kimaldi.com/

[8] http://www.enttia.com

[9] http://www.identtia.com

[10] http://stackoverflow.com

[11] http://www.grupospec.com/

[12] http://www.robotics.es/

[13] http://www.webdesignerdepot.com/2012/06/ui-vs-ux-whats-the-difference/

[14] https://msdn.microsoft.com/en-us/library/8bs2ecf4%28v=vs.110%29.aspx

[15] http://www.software-architects.com/devblog/2013/10/17/AngularJS-with-TypeScript-and-Windows-Azure-Mobile-Services

[16] http://en.wikipedia.org/wiki/Usage_share_of_web_browsers#Summary_table

[17] http://www.w3schools.com/browsers/browsers_stats.asp

## 10.3 Taula de figures

Il·lustració 1 – Terminals del fabricant Suprema ..... 10
Il·lustració 2 – Terminals del fabricant Nitgen ..... 10
Il·lustració 3 – Logos d' Enttia i Identtia ..... 14
Il·lustració 4 – Infraestructura física IDPresence ..... 15
Il·lustració 5 – Estructura física Terminal Manager ..... 16
Il·lustració 6 – IDPresence actual ..... 18
Il·lustració 7 – Diferències de funcionalitat IDPresence ..... 21
Il·lustració 8 – Pipeline de MVC.net ..... 24
Il·lustració 9 – Funcionament HTML ..... 26
Il·lustració 10 – Funcionament Web 2.0 ..... 26
Il·lustració 11 – Funcionament SPA ..... 27
Il·lustració 12 – Patró MVC (Model – View – Controller) ..... 28
Il·lustració 13 – Sensacions amb Angular JS ..... 28
Il·lustració 14 – Arquitectura de 3 capes (3-Tier-Layer) ..... 32
Il·lustració 15 – Esquema Seu – Edifici – Planta ..... 34
Il·lustració 16 – Esquema franges horaries ..... 35
Il·lustració 17 – Explicació compilació .net ..... 46

## Resum

Aquest projecte tracta de la realització d'una migració tecnològica (en aquest cas una actualització de software) a un programa de Gestió de Control d'Accés i Presència. S'ha dut a terme un estudi de què és una migració tecnològica. S'evoluciona un software (IDPresence) per poder aconseguir una versió millorada que supleixi les mancances de la versió original. S'ha implementat una solució al problema duent a terme tots els passos: anàlisi de la problemàtica i requeriments necessaris, implementació d'una solució robusta, moderna i de fàcil manteniment i d'una fase de test final per tal de garantir una qualitat i la funcionalitat necessària per un producte professional. Finalment, s'ha implantat el sistema en diversos clients per tal de valorar la solució i extreure opinions i els resultats de la migració.

## Resumen

Este proyecto trata de la realización de una migración tecnológica (en este caso una actualización de software) a un programa de Gestión de Control de Acceso y Presencia. Se ha llevado a cabo un estudio de que es una migración tecnológica i se ha evolucionado el software (IDPresence) para poder conseguir una versión mejorada y que supla las carencias de la versión original. Se ha implementado una solución al problema llevando a cabo todos los pasos: análisis de la problemática y requisitos necesarios, implementación de una solución robusta, moderna i de fácil mantenimiento y una fase final de test para garantizar una calidad y una funcionalidad necesaria para un producto profesional. Finalmente, se ha implantado el sistema en distintos clientes para valorar la solución y extraer los resultados de la migración.

## Abstract

This Project is about the realization of a technological migration (in this case, a software upgrade) to an Access and Presence Managing Application. There has been an analysis of what is a Technological Migration and a software upgrade of IDPresence to achieve an improved version make up for the shortcomings of the original version. We have implemented a solution doing all the steps involved in the process: analysis of the problems and required requirements necessaries to implement a robust, modern and attractive solution with easy maintenance and a final test phase to assure the quality and functionality of a professional solution. Finally, the system has been tested into clients' environments to evaluate the solution and extract opinions and results from the migration.