

## **Treball final de grau**

**Estudi:** Grau en Enginyeria Informàtica

**Títol:** **Software d'integració global d'escaneig  
i gestió documental**

**Document:** Memòria

**Alumne:** Francesc Puig Plana

**Tutor:** Josep Soler Masó

**Departament:** IMAE

**Àrea:** LSI

**Convocatòria (mes/any):** Juny 2017



---

## Continguts:

---

<b>1</b>	<b>Introducció, motivacions, propòsit i objectius del projecte</b>	<b>1</b>
1.1	Introducció . . . . .	1
1.2	Motivacions i propòsit . . . . .	3
1.3	L'Empresa . . . . .	3
1.4	Objectius . . . . .	4
1.5	Punt de partida . . . . .	7
<b>2</b>	<b>Estudi de viabilitat</b>	<b>9</b>
2.1	Recursos necessaris . . . . .	10
2.2	Viabilitat tecnològica i econòmica . . . . .	10
<b>3</b>	<b>Metodologia</b>	<b>11</b>
3.1	Treball amb iteracions . . . . .	12
<b>4</b>	<b>Planificació</b>	<b>15</b>
4.1	Aclariments previs . . . . .	15
4.2	Tasques proposades . . . . .	15
4.3	Cronograma temporal . . . . .	17
4.4	Fases de treball . . . . .	19
4.5	Valoració de la planificació . . . . .	23
4.6	Freqüència de desenvolupament . . . . .	24
<b>5</b>	<b>Marc de treball i conceptes previs</b>	<b>25</b>
5.1	ERP . . . . .	25
5.2	Base de Dades . . . . .	26
5.3	Entorn bàsic de desenvolupament . . . . .	26
5.4	Sistema a desenvolupar . . . . .	27
5.5	Marc de treball per desenvolupar un programa d'escriptori . . . . .	28
5.6	Marc de treball per desplegar una aplicació web . . . . .	29
5.7	Nom comercial ScannerApp . . . . .	30
<b>6</b>	<b>Requisits del sistema</b>	<b>31</b>
6.1	Iniciar el software des de l'ERP . . . . .	31

---

6.2	Classificació de documents . . . . .	31
6.3	Escàners disponibles . . . . .	32
6.4	Preferències . . . . .	32
6.5	Digitalització Nativa . . . . .	32
6.6	Digitalització . . . . .	33
6.7	Perfils . . . . .	33
6.8	Edició i previsualització . . . . .	33
6.9	Tractament i selecció . . . . .	34
6.10	Systray Icon . . . . .	34
6.11	Requisits no funcionals . . . . .	35
<b>7</b>	<b>Estudis i decisions</b>	<b>39</b>
7.1	Requisits segons el sistema de l'empresa . . . . .	39
7.2	El perquè de Python . . . . .	39
7.3	Accés a Escànners . . . . .	40
7.4	PyQt . . . . .	47
7.5	ØMQ . . . . .	49
7.6	SQLite . . . . .	51
7.7	cx_Freeze . . . . .	52
7.8	Diagrama bàsic de funcionament . . . . .	55
<b>8</b>	<b>Anàlisi i disseny del sistema</b>	<b>57</b>
8.1	Estructura modular . . . . .	57
8.2	Anàlisi de Responsabilitats . . . . .	57
8.3	Disseny de l'API . . . . .	59
8.4	Diagrama de casos d'ús . . . . .	61
8.5	Diagrama de classes bàsic . . . . .	63
8.6	Diagrama de classes principal . . . . .	64
8.7	Diagrama de seqüència Llistar Escànners . . . . .	65
8.8	Diagrama de seqüència Request . . . . .	66
8.9	Diagrama de seqüència Crear perfil . . . . .	67
8.10	Diagrama de seqüència Obtenir document . . . . .	68
8.11	Assignació de responsabilitats . . . . .	69
8.12	Models de dades . . . . .	70
<b>9</b>	<b>Implementació i proves</b>	<b>73</b>
9.1	Adaptació sockets natius de Python amb sockets ZMQ . . . . .	73
9.2	Twain . . . . .	74
9.3	Visors PDF . . . . .	74
9.4	Instal·lador . . . . .	77
9.5	Accés directe del software automàticament . . . . .	78
9.6	Compiled Version vs Uncompiled Version . . . . .	78
9.7	Navegació ERP . . . . .	80
<b>10</b>	<b>Implantació i resultats</b>	<b>81</b>
10.1	Integració Sentry i Logger . . . . .	81
10.2	Sockets configurables . . . . .	82
10.3	GUI . . . . .	82

10.4 Client . . . . .	94
10.5 API . . . . .	95
10.6 Connector . . . . .	98
10.7 Systray . . . . .	100
10.8 Threading . . . . .	101
10.9 ScannerAgent . . . . .	102
10.10 Protocol Twain . . . . .	104
10.11 Integració amb un ERP . . . . .	110
10.12 Control d'errors . . . . .	113
10.13 Resultats i proves . . . . .	113
<b>11 Treball futur</b>	<b>129</b>
<b>12 Conclusions</b>	<b>131</b>
<b>13 Bibliografia</b>	<b>133</b>



# CAPÍTOL 1

---

## Introducció, motivacions, propòsit i objectius del projecte

---

### 1.1 Introducció

Cada cop és més necessari la utilització de documents en format digital, amb tot el que això comporta. Digitalitzar, emmagatzemar, distribuir, ordenar, consultar i modificar un gran volum de documents pot arribar a ser una tasca tediosa.

Això fa que moltes empreses es proposin buscar una solució per automatitzar aquests processos. El primer pas és comptar amb una base informàtica, i després plantejar com organitzar aquests documents. Existeixen diverses solucions software, i una bona base és la metodologia que proposen els gestors documentals. Un gestor documental intenta integrar totes aquestes característiques i fer molt més fàcil els processos que respecten al tractament dels fitxers, automatitzant les tasques típiques d'aquests, tant l'obtenció del document en format digital, com l'edició i la postclassificació. Automatitzant tasques es pot arribar a aconseguir un procés molt més àgil. En països hispans s'utilitza el terme gestió documental, tot i que comptant amb les característiques actuals de les empreses és més correcte utilitzar el terme anglosaxó *Enterprise Content Manager -ECM-*, atès que les empreses utilitzen un ampli ventall de tipus d'arxius digitals.

De la necessitat de contribuir amb el medi ambient i el maldecap que suposa l'emmagatzemament en format paper dels documents, ha sorgit la metodologia **paperless**, que concretament proposa la unificació de dades digitalitzades i l'estalvi de documents en format paper. Ja no només es parla d'estalvi de paper, sinó de classificació i espai, que pot arribar a ser un verdader maldecap. Però amb tal volum de dades digitalitzades cal remarcar que l'organització hi juga un paper clau, i és aquí on entra el rol de la gestió documental. Si es pensa i s'analitza la quantitat de documents que genera tan sols una petita/mitjana empresa en el seu dia a dia, de seguida sorgeix la necessitat d'una bona base per gestionar-ho, i no tan sols per l'organització, sinó per l'obtenció, visualització... d'aquests documents. Això no vol dir que no s'estigui preparat, però altres metodologies com aquesta fan clar l'estalvi de temps, recursos... i això obviament es pot traduir en guanys econòmics. Parlant d'estar preparats, qualsevol pot tenir el seu sistema, però subjectivament sóc partidari d'indagar amb la nova tecnologia, que no només ha de servir per tenir una vida quotidiana plena d'avantatges i/o facilitats. Al primer graó de la tecnologia vanguardista hi ha d'haver aquella que ajudi al

progrés de les empreses.

Anàlogament un sistema de gestió documental permet augmentar l'eficiència de treball automatitzant i agilitzant tasques molt comunes del dia a dia en la gran majoria d'empreses, en el que el tractament de documents es refereix. Observant l'esmentat anteriorment, són moltes les funcionalitats que pot tenir un software d'aquestes característiques, l'aplicació pot integrar processos de reconeixement de fitxers, visualitzadors per categoria, analitzadors de malware o documents erronis, processos post classificació... Bàsicament ajuden a l'usuari a gestionar aquests processos, segons la necessitat de cadascú, i afegeix intel·ligència per tal de classificar, comprovar... o el que sigui necessari amb aquesta base de documents.

Per tal de donar una cara més familiar a un gestor documental, es pot definir en paraules triviais com un *contenido* de documents intel·ligent, que no només ajuda amb l'obtenció del fitxer (foto, vídeo, document de text pla...), sinó que t'organitza en directoris, seccions... (el que es vulgui), aquests documents. A més se li poden aplicar les característiques que es vulguin (edició de documents, filtratge antimalware, màscares de documents...), i proporciona una base classificada de tots aquests arxius. A tall d'exemple, es podria veure com una biblioteca ben organitzada de documents amb un empleat que no tan sols organitza i posa a disposició de qui ho necessiti certs fitxers, sinó que ajuda i automatitza la mateixa obtenció dels esmentats.

Per posar color a la temàtica, ara suposem una tasca que realitzaria un empleat d'una empresa x per digitalitzar i adjuntar una factura d'un cert client, sense cap base gestió documental. Pensem que el client es persona a la suposada empresa, juntament amb més clients que esperen impacients el seu torn. L'empleat escaneja el document, l'aplicació demana on desar-lo, el desa en qualsevol directori del pc físic, busca el programa, l'obre, busca el client, es dirigeix a l'apartat de factures, busca la referència, selecciona *adjuntar factura*, busca en el directori la factura que acaba de digitalitzar, la selecciona i l'ajunta al programa. Ara suposem que aquest empleat té un software d'integració i gestió documental: Obre el programa, cerca el client i digitalitza el document. El mateix software ha aconseguit digitalitzar i classificar la factura respectiva al client seleccionat, no calen més passos i ja pot atendre al següent client. Vegem-ho en forma de diagrama.

Cas 1



Cas 2



El procés de digitalització de documents és fonamental per representar digitalment el mapa de bits d'un document en format paper, mitjançant aquest procés s'emmagatzema en una base de dades a la qual tindran

accés les persones autoritzades de l'arxiu o informació capturada. Al final el procés sempre és el mateix i es basa en convertir un origen no digital a una representació digital. El gran embroll de tot això és l'organització, si en format paper és complicat, en format digital també pot suposar un problema, ja que no només és necessari un espai en disc sinó que calen mètodes d'organització i accés.

La necessitat d'aquest tipus de software, la proposta feta pels mentors i les ganes d'indagar en una metodologia desconeguda han estat claus per l'elecció d'aquest projecte. Tot discutint mentalment en què dedicaria les hores en els següents mesos, en els que em tocava fer el projecte, tenia clar que volia fer quelcom que m'apassionés, que m'aportés coneixement i motivació. Tenia diverses idees mig difuminades quan em varen donar l'oportunitat de realitzar aquest projecte, no vaig dubtar. Al veure les cares dels qui m'ho proposaven vaig veure que era un projecte necessari, i interessant. Mai no he cregut en els projectes de fons d'armari, que acaben oblidats sota un munt de folis.

## **1.2 Motivacions i propòsit**

Com a opinió personal em reservo el dret de dir que qualsevol projecte o treball a realitzar és molt important que hom se'l faci seu, que el senti tangible i que sobretot cregui en la importància d'aquest. D'aquesta manera es torna motivador realitzar-lo, i sempre es rep un feedback positiu quan és així. Clar està que treballant a gust fa que tot es torni més de cara. D'aquesta manera les coses surten millor, el projecte avança fluidament i a més a canvi n'obtens una satisfacció, no només per estar o haver realitzat un projecte o treball sinó d'estar-ne aprofitant la avinentesa per gaudir i créixer en l'àmbit de treball en què es realitza. Grans empreses conegudes com Google utilitzen mètodes de treball motivadors per als seus empleats al·legant que un treballador content realitza un volum més ampli de feina i de més qualitat. Els beneficis de treballar així no tan sols es veuen reflectits en el producte final sinó que ajuden a fer passional allò que s'està fent en cada moment del qual dura el procés. Per tant, invertir una mica més de temps en buscar un bon tema, una bona proposta, un bon entorn de treball, i la motivació, són claus per aconseguir tot el relatat. A més, com a extra està bé aprofitar l'ocasió per escollir metodologies i estils de treball que sempre havies volgut provar, donant més força així a la motivació del treball.

Anecdòticament el dia que vaig tenir clar el tema, vaig tenir una conversa on sense jo explicar sobre el que volia fer el TFG, em van expressar el necessari que seria un software d'aquestes característiques, comentant un tema relacionat amb l'advocacia, va ser llavors quan em va fer il·lusió comentar que precisament el treball que començava tractava del mateix.

Per situar el lector en el marc d'aquest treball, aquest TFG es realitza a l'empresa GISCE-TI, S.L. Un client havia demanat un software d'integració d'escaneig i gestió documental. En capítols posteriors es detallaran més a fons les característiques i el treball empresa - client, però és interessant deixar clar que aquest projecte es desenvolupa a l'empresa per servir el producte al client que l'ha sol·licitat. Per tant ja està "col·locat" abans de presentar-lo, ja que es desenvoluparà per al client.

## **1.3 L'Empresa**

El projecte es desenvolupa a l'empresa, i a fi de presentar la situació actual, es tracta breument les característiques d'aquesta. L'empresa on es realitza aquest TFG és GISCE-TI, S.L. una empresa Gironina especialitzada en servir solucions software a les empreses del sector elèctric, tant distribuïdors com comercialitzadores. A GISCE, hi treballa un equip de programadors per assolir i desenvolupar aquestes solucions

software. El producte estrella és un ERP integral que ajuda a satisfer les necessitats d'aquests tipus d'empreses. També hi ha el departament d'Enginyeria que es dedica a inspeccions, auditòries...

Aquest TFG tracta de desenvolupar una solució software sol·licitada per una empresa client, que es pugui integrar també en el treball funcional i diari d'aquest ERP, i que més endavant hi hagi la possibilitat de desplegar-ho a altres clients.

Els productes que serveix l'empresa, són Software lliure, i la filosofia de treball que utilitzen és donar suport a les empreses que sol·liciten els productes, a nivell usuari o a nivell informàtic. També ajuden a l'automatització de tasques administratives o legislatives dels clients...

Al ser Software lliure, el software es pot desplegar on es vulgui i per qui vulgui, però l'empresa es nodeix de donar suport, dels molts casos que surten, els nous desplegaments, actualitzacions, canvis en el software sota demanda del client (a mida)...

Aquest apartat no busca descriure extensament com és i com treballa aquesta empresa, ja que queda fora de l'àbast d'aquest document, sinó donar a conèixer l'entorn actual on es desenvolupa el TFG per entrar una mica més en la temàtica i poder entendre més el perquè de certs conceptes.

### **1.4 Objectius**

L'objectiu d'aquest TFG és desenvolupar l'anàlisi, disseny i implementació d'un sistema que permeti la gestió documental, aprofitant l'oportunitat per iniciar-se i aprendre altres metodologies de treball, nous entorns i noves característiques. En definitiva no limitar-se a marcar un sol objectiu de finalitzar i entregar un producte final, sinó també buscar altres objectius per aprendre, com per exemple: nous mètodes d'organització de la feina, noves característiques dels llenguatges de programació o fins i tot generació de la documentació per a la memòria. Aquestes petites fites són les següents:

- Metodologia Scrum
- Documentation Generator
- Python Cheat Sheet
- React Introduction
- Tests
- Free Software

Entrant més en detall en aquests punts, a continuació es detalla cadascun:

#### **1.4.1 Scrum**

A l'hora d'organitzar la feina, em marco com a objectiu aprendre a treballar amb metodologies scrum, que en capítols posteriors es detallaran millor, però és un procés que s'utilitzen de forma regular les bones pràctiques per treballar col·laborativament, en equip i obtenir el millor resultat possible pel projecte. S'aprofitarà per aprendre a treballar amb metodologies iteratives de tasques.

#### **1.4.2 Documentation Generator**

El fet d'haver de redactar una memòria també es pot aprofitar per aprendre noves tècniques per generar documents. Els documentation generator són precisament això, auto generadors de documents que converteixen fitxers en un format concret en HTML, PDF o altres formats. Interpreten el codi font que ha escrit l'usuari i reconeixen les marques pròpies del llenguatge de marques que s'està utilitzant per generar automàticament la documentació. Els avantatges són que es poden utilitzar automatismes ràpids per escriure en notació matemàtica, crear gràfiques, inserció d'imatges, creació de títols, indexs automàtics i un munt d'avantatges més. En concret s'aprendrà i s'utilitzarà *Sphinx* que utilitza *ReStructuredText* com a llenguatge de marques lleuger.

#### **1.4.3 Python Cheat Sheet**

Fins ara havia treballat molt poc amb *Python2.7*, i a l'iniciar un projecte nou, és una bona pensada canviar i començar a utilitzar la versió de Python més nova que hi ha en el mercat (actualment *Python3.6*). Per tant l'objectiu és aprofundir en les novetats d'aquesta versió i aprendre els *cheat sheets* de Python. En aquest mateix apartat també apareix un nou objectiu, i és utilitzar més una programació amb consciència i conseqüència, és a dir, invertir més temps en deixar una línia de codi més clara i fàcil de reimplementar, que no deixar-la funcionant però que es pugui millorar molt més. Per sort el llenguatge ho permet, ja que és molt polivalent i compta amb moltes eines per ajudar al programador a escriure un codi funcional, lleible i perquè no divertit. De fet el seu epònim és Monty Python perquè en els orígens, el seu creador Guido van Rossum va voler crear un llenguatge divertit i fàcil d'utilitzar, fent honor així a una sèrie d'humor del qual era fan el van Rossum, els Monty Python.

#### **1.4.4 React Introduction**

Si es té temps, s'intentarà desplegar un “visor” web per gestionar la base del gestor documental, bàsicament per aconseguir navegar entre els documents digitalitzats i classificats en una base informàtica. Com que aquest projecte estarà casat en l'entorn de treball d'una empresa, de moment aquest visualitzador s'intentarà adaptar a la base informàtica d'aquesta. Per fer això s'utilitzarà una infraestructura que no he utilitzat mai, React js, per tant com a objectiu si hi ha temps, és aprendre a desplegar una aplicació web en aquesta llibreria.

#### **1.4.5 Tests**

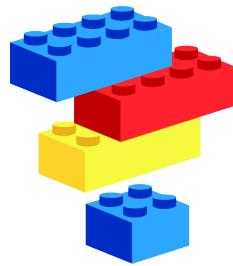
És d'important incòmpanya pel programador realitzar el màxim de proves i tests si es vol que el software sigui estable, pugui anar corrent i escalant versions, i que sigui fàcil de mantenir. Però hi ha moltes i moltes maneres de testejar un software, i com a objectiu em marco provar-ne una de la qual desconeix però em sembla interessant, el TDD: Test-Driven Development, una tècnica de disseny de software que consisteix en escriure primer les proves, després que el codi passi una validació d'aquestes proves i finalment refactoritzarlo. Aconseguint un codi molt més robust, mantenible i resistent a fallades.

### **1.4.6 Anàlisi, disseny, i implementació**

L'objectiu en si d'aquest TFG és dissenyar, implementar i desplegar una plataforma de digitalització (escaneig) i un mòdul de gestió documental. Queden en segon pla per tant la visualització d'aquesta gestió mitjançant una aplicació web. En aquest mateix apartat se'n detallen més endavant les característiques d'aquest projecte, per definir més bé quins són els objectius marcats en un principi.

Dins aquest objectiu coexisten subobjectius o objectius més atòmics, a tall d'exemple no sols es desenvoluparà l'aplicació final i punt, sinó que s'intentarà anar més enllà i investigar el que realment pot fer diferent i atractiu el producte. Ja d'entrada donant un cop d'ull a projectes similars, o software de mercat, no n'hi ha cap que sigui “independent”, que sigui scalable, modelable, adaptable, i per tant s'intentarà construir una infraestructura totalment scalable, modelable i que es pugui utilitzar de forma independent al sistema. És obvi que ja existeixen alguns softwares d'integració global de digitalització i gestió documental, per tant la proposta és realitzar quelcom nou, desenvolupar un software que es pugui utilitzar en diferents infraestructures i cadascuna de les parts del software (mòduls) sigui independent. Per tant l'objectiu és la creació de mòduls independents, per tal de poder utilitzar-los tots o només els que es desitgi. A més que sigui fàcil i àgil d'adaptar a altres metodologies si es vol canviar la comunicació, l'accés a la base de dades...

A grans trets s'hauria de poder aprofitar la part que es vulgui del software per altres propòsits, comunicacions... Si s'aconsegueix fer petits mòduls de cada apartat o característica de l'aplicació final, queda un producte ensamblat com si cada mòdul fos una peça de construcció. D'aquesta manera es pot intercanviar qualsevol d'aquestes peces, o simplement utilitzar-les de forma unitària.



### **1.4.7 Free Software**

Més que un objectiu, una metodologia, el free software s'adapta totalment al tipus de software que es pretén desplegar, per les seves característiques i per la filosofia ben marcada que defineix l'empresa (GISCE-TI) on es desenvolupa aquesta aplicació.

- Característiques d'aplicació:

En aquest mateix apartat es senyala la importància de desplegar un tipus d'infraestructura que no es trobi al mercat. El que ho fa especial és que no sigui un software dependent i que es pugui adaptar molt fàcilment. El fet que sigui reutilitzable per qualsevol programador fa necessari aplicar aquesta filosofia, per tal de fer-lo accessible i “modelable” a gust de tothom. Si el software es torna adaptable, però per temes legals que marca la llicència de software no es pot modificar, no s'obté cap guany en aquest aspecte.

- Filosofia i carisma de l'empresa:

És una empresa que sempre ha treballat amb Free Software, de fet, tant les eines de *development* com el software que serveixen és tot Software lliure.

Per tant un dels objectius també és aprendre les característiques, la utilització i distribució d'aquests tipus de llicència.

Aquest software utilitza llicència MIT

## **1.5 Punt de partida**

Havent donat a conèixer l'entorn de l'empresa on es desenvolupa el TFG, com a concepte previ als capítols que prossegueixen, és important mencionar la base que inicia el projecte, i quins en podran ser els punts claus. Es demana per tant que el software a desenvolupar pugui anar integrat amb l'ERP, però alhora que treballi de manera independent.

Inicialment es parteix de la base que aquest ERP està muntat en l'entorn de treball i es podrà començar a treballar amb el projecte de manera independent, fins a arribar al punt d'integració, que és on es necessitarà afegir funcionalitats a l'ERP per tal de treballar amb el software. En quant al programa, no hi ha res fet, excepte unes proves de concepte que es van fer en el seu dia, però cal decidir, dissenyar, engregar i desplegar tot l'entorn i funcionalitats per desenvolupar de 0 el software.

Aquest ERP utilitza un sistema d'adjunts per adjuntar en cada recurs d'aquest. Per exemple una pòlissa pot tenir adjuntades factures, un subministrament pot tenir adjuntats els BOEs corresponents amb les normatives i canvis... Això ja està fet, el que es pretén és desplegar tot el sistema de digitalització i obtenció d'aquests adjunts així com la classificació i procés d'adjuntar de forma intel·ligent i automàtica, així com altres temes tractats i exposats ja en aquest treball.



## CAPÍTOL 2

---

### Estudi de viabilitat

---

Un client de l'empresa on estic fent aquest TFG va sol·licitar un gestor documental, els hi era molt necessari per a estalviar temps. Concretament quan els hi arribaven varis clients alhora, i quan havien de digitalitzar un document i adjuntar-lo a la secció corresponent de l'ERP tardaven molt, ja que el procés d'escanejar, guardar, cercar la secció a l'ERP, anar a buscar-lo i adjuntar-lo és llarg i quan es torna repetitiu, automatitzar aquest procés pot significar un estalvi de temps molt gran. A més desenvolupar-ne un per un client, significa que els altres amb un grau molt alt de probabilitat, també el voldran, i es podrà comercialitzar i desplegar amb molts altres clients. Al ser un mòdul apart de l'ERP, és opcional, però ja que hi ha una integració directa amb els clients, es podria canviar la paraula opcional per “obligat” perfectament.

Cal subratollar que aquest TFG engloba una part del desenvolupament d'aquest software. Realment el que es demana des de la meva empresa és que es pugui continuar amb l'ampliació de noves característiques que s'escapen a l'abast d'aquest treball. Si es pretén un enfoc molt més genèric sobre la viabilitat actual d'un software d'aquestes característiques, es poden tenir en compte dos punts importants:

1. Preocupació i ocupació: Tal com s'introdueix en apartats anteriors, un problema vital de moltes empreses és la generació i posterior tractament/classificació dels documents. Senzillament una necessitat crea una ocupació, per tant es pot parlar de preocupació-ocupació. Això fa evident que la viabilitat d'aquest projecte és directament proporcional a la necessitat de l'esmentat en el mercat d'empreses.
2. Necessitat en el mercat: Si es revisa el software que hi ha d'aquestes característiques, en trobem de dos tipus.
  1. De pagament i tancats
  2. Desenvolupat i utilitzat única i exclusivament per una empresa

Realment donant una ullada per internet i estudiant el mercat actual d'aquest tipus de software, hom s'adona que tan si s'enfoca des de la vessant directiva, com del departament de IT d'una empresa, les limitacions són moltes i les solucions poques. Si una petita/mitjana empresa no compta amb departament de IT, ha de comprar o encarregar a mida un software d'aquestes

característiques. I si el té, els desenvolupadors han de començar pràcticament de 0 existint molt pocs mòduls per fer aquest tipus de feina, i que realment s'ha de fer una adaptació. Per tant el que planteja aquest projecte és unificar això per tal de:

1. S'utilitza a nivell usuari: una empresa contracta aquest software
2. A nivell de programació: un programador podrà fer el que vulgui amb l'aplicació:
  - Desplegar-la íntegrament
  - Utilitzar certs mòduls de l'aplicació *casant-los* amb els seus (la comunicació per exemple)
  - Contribuir en canvis de l'aplicació
  - Tot això distribuït amb software lliure

### **2.1 Recursos necessaris**

A part de les eines de desenvolupament i una màquina física per desenvolupar el software, és necessari un escàner per digitalitzar documents.

### **2.2 Viabilitat tecnològica i econòmica**

Subjectivament aquest producte es ven sol, ja que l'ha solicitat el client, probablement s'aplicarà als altres, i que només cal veure l'atenció tant per part dels mentors a l'empresa, com els clients, que realment demostren atenció i seguiment de com evoluciona la fase de desenvolupament d'aquest software. “Ho estan esperant amb candeletes”. Apuntant en un enfocament de futur i durabilitat, per analitzar bé la viabilitat tecnològica que pot tenir aquest software, es pot veure que serà un software en creixement continuu. Sempre s'hi podran afegir coses noves, sigui per iniciativa de l'empresa o del client. A tall d'exemple pensem que les possibilitats són tantes com fins on ens voli la imaginació: filtres OCR, protecció de documents, antivirus de documents, seguretat, consultes des d'un smartphone, i un llarg etc. Per tant es pot considerar que tecnològica i econòmicament és més que viable.

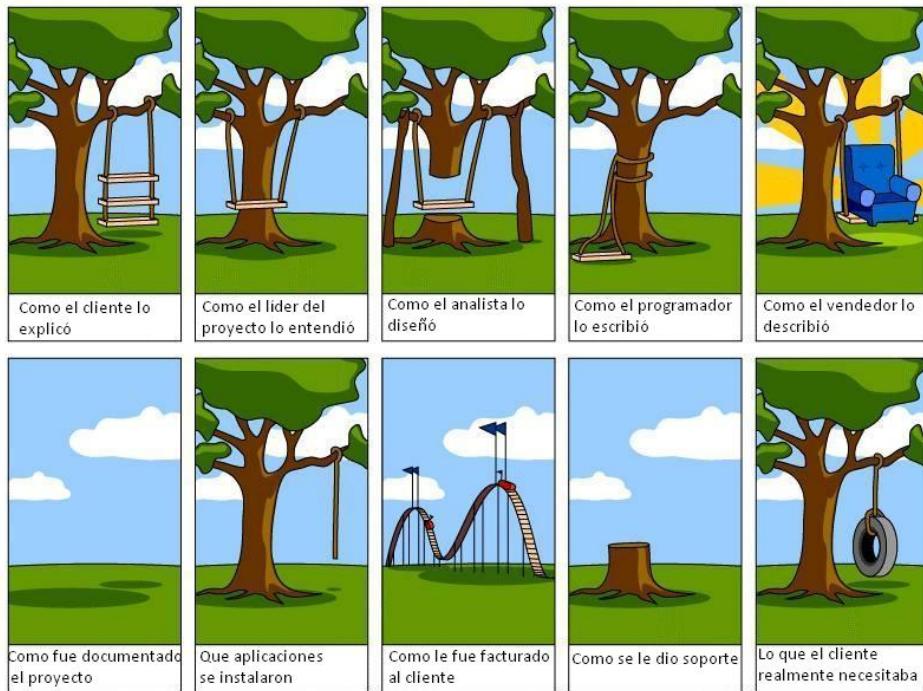
# CAPÍTOL 3

---

## Metodologia

---

Les metodologies Ágil són una de les metodologies més utilitzades a l'hora de treballar en el seguiment d'un desenvolupament de software. Es proposa treballar amb iteracions per setmana o un temps establert, a més durant el desenvolupament del TFG es mantindrà un feedback directe amb el client per tal de portar un control sobre el software. També hi haurà versionat de software cada 'x' temps, sortirà una versió alpha/beta del programa per desplegar i fer una demo de cara al client. A partir d'aquestes demos, poden anar sortint millors i modificacions no contemplades inicialment. Aquest és un bon mètode de treball per resoldre petits malentesos de captar una idea inicial dels requisits d'un software. A més aquest feedback amb el client també serà visible pels membres de l'empresa, per poder estar alerta dels canvis i de l'evolució de l'aplicació. Una de les primeres coses que s'ensenyen en docència d'Enginyeria del Software és la següent:



Aquest tipus de metodologia i el treball colze a colze amb el client ajuden a solucionar aquests problemes. A poc a poc es pot construir un software fidel a la idea del client i a la visió del desenvolupador.

### 3.1 Treball amb iteracions

Relacionat amb el treball d'iteracions, ajuda molt més a portar un seguiment fidedigna al producte final. Si s'intenta complir els requisits que marquen aquest tipus de metodologies, s'obté una ajuda molt bona a l'hora de construir una aplicació de manera incremental. Si no, es pot tornar un *caos* i seria com començar la casa pel teulat.

En aquest TFG s'ha seguit el següent: Aquest tipus de treball consisteix en un desenvolupament iteratiu i incremental, on els requisits i solucions evolucionen segons la necessitat del projecte. El desenvolupament iteratiu i incremental marca la creació d'iteracions o *sprints* per realitzar una sèrie de tasques en cadascuna d'aquestes iteracions o *sprints*. Les iteracions o *sprints* consten d'un temps predeterminat fixat anteriorment, i que el/els desenvolupadors han d'intentar complir exhaustivament. Aquestes tasques com es menciona anteriorment es fixen amb consciència i coneixement del desenvolupament, és important fixar les tasques de manera ordenada, incremental i seguint una estructura lògica. Com és obvi durant la fase de desenvolupament s'aniran afegint i tancant tasques.

Cada iteració inclou: planificació, anàlisi dels requisits, disseny, codificació, tests i documentació. És de vital importància el concepte de “Finalitzat” (*Done*), ja que la idea és afegir funcionalitats al software i deixar-lo estable. Per tant una tasca única i exclusivament es passarà a *Done*, si està realment acabada, provada i funcional.

En resum es treballarà amb iteracions per la realització de tasques, es farà servir un sistema Scrum per mantenir un feedback millor amb el client i un sistema de versionat per tal d'avançar entre versions (releases).

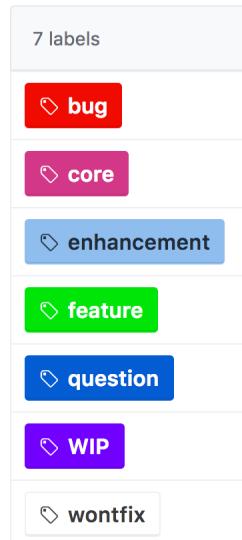
A més a més, s'han utilitzat paràmetres que ajuden més a la identificació, organització, realització... de les

tasques:

- Labels: s'assignaran etiquetes a les tasques, per tenir una relació de quin tipus de tasca es tracta, a que afecta... :
  - Bug
  - Core
  - Enhancement
  - Feature
  - WIP
  - Question
  - Wontfix
- Issue / PR: sistema de creació d'issue i Pull Request que la tanqui
- Branches: cada funcionalitat nova es desenvoluparà paral·lelament en una o més branques de git
- Milestones: S'associen a les iteracions
- Assignes: Responsable de la tasca (per si s'amplia el projecte)

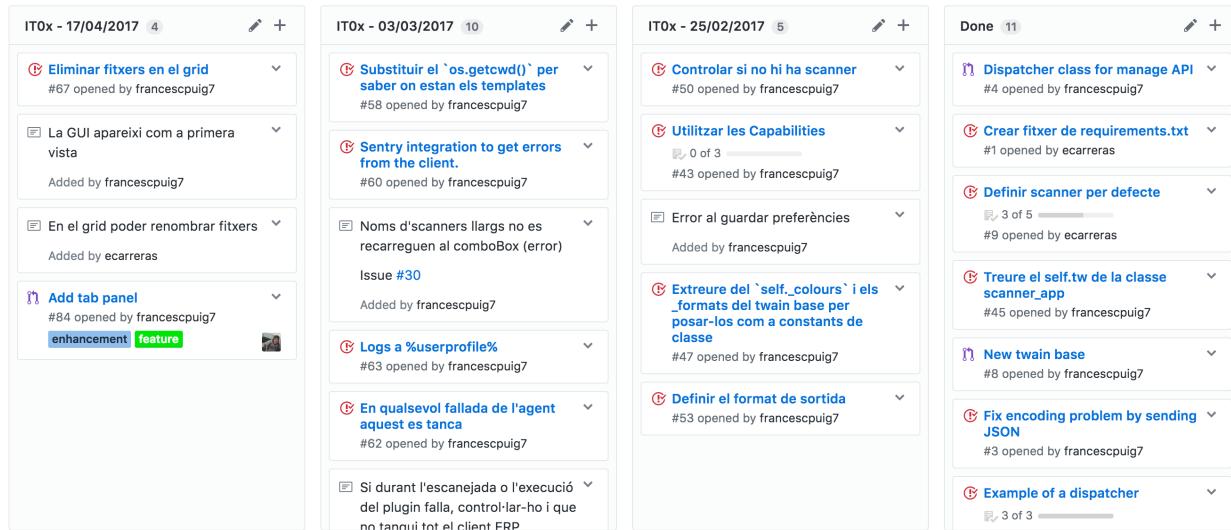
### **3.1.1 Labels**

La següent figura mostra els labels utilitzats per controlar les Issues i les PR del projecte:



### 3.1.2 Iteracions

La imatge que es mostra a continuació mostra la pissarra amb les tasques que es van realitzant dia a dia:



A continuació es llisten les eines que s'utilitzaran per fer tot això:

- Treball d'iteracions GitHub: <https://github.com/>
- Versionat GitHub: <https://github.com/>
- Scrum Trello: <https://trello.com/>

# CAPÍTOL 4

---

## Planificació

---

### 4.1 Aclariments previs

La mitjana de dedicació ha estat en unes 4 hores diàries 5 dies a la setmana des del mes de Gener.

Molt temps ha estat invertit en l'aprenentatge de característiques de la llibreria per digitalitzar documents i en modificacions del software.

La planificació s'ha fet mitjançant una data d'entrega i dividint les iteracions en una certa durada estableguda. Genèricament les iteracions es comptava que serien d'una setmana, però per dedicació, volum de feina i molts més aspectes influents es poden haver allargat més, definit-ho així a l'apartat de temporització.

### 4.2 Tasques proposades

A continuació es proposa una llista de tasques a realitzar per la realització del projecte:

- Anàlisi de requeriments
- Models de treball
- Estructura esquemàtica del projecte
- Creació dels models de treball
- Definició diagrames de casos d'ús, classe i seqüència
- Estudi de llibreries segons els requeriments de l'aplicació
- Disseny de l'ús de les llibreries
- Disseny d'activitats i tasques

- Definir projecte scrum
- Instal·lació de l'entorn de treball
- Creació del repositori
- Definició de la llicència de software
- Definició de l'estructura de classes del projecte
- Estudi d'entorn gràfic i modelats d'accés als recursos
- Definir estructura de dades
- Anàlisi de casos d'ús per un primer disseny d'interfície d'usuari
- Definició d'interfície d'usuari
- Definició de la comunicació
- Estudi de llibreria per sockets
- Definició i utilització de llibreria per comunicació amb escàners
- Implementació de les primeres classes
- Disseny de l'API
- Implementació de la comunicació
- Definir model E/R de dades
- Estudi de llibreria per la creació de la base de dades
- Creació de la base de dades
- Obtenir document amb escàner
- Obtenir detalls del document obtingut
- Implementació de les preferències d'escaneig
- Guardar a la base de dades
- Protocol base de Source Manager
- Definir paths del projecte
- Creació i desenvolupament base del sistema
- Conversors de documents
- Desenvolupament de l'interfície d'usuari
- Desenvolupament d'un client de la vista
- Desenvolupament de la comunicació
- Desenvolupament de l'API
- Definició del protocol Source Manager
- Implementació del protocol Source Manager

- Definició del plugin a ERP
- Creació del plugin a l'ERP
- Estudi de llibreria per executable i instal·lador
- Definició de les dependències pel setup
- Creació del setup
- Preparació de versió
- Tests
- Versió estable

Per seguir un bon procés de treball, és bo que cada funcionalitat quan es fan les tasques de desenvolupament es faci un procés de tests i desplegament. De fet es treballa amb un repositori a github, i s'estructura el treball amb branques. La filosofia a seguir és que una funcionalitat no entri a màster fins que no s'hagi desenvolupat i testejat exhaustivament. Es podria desenvolupar una funcionalitat i incloure-la al master del projecte es té la percepció de que funciona, posteriorment fer una versió i testejar totes les funcionalitats, però per estalviar problemes i una feina posterior a esbrinar d'on prové l'errada, totes les tasques de desenvolupament constaran de 2 o 3 etapes segons si s'ha de desplegar a casa el client:

- Desenvolupament
- Test
- Versió i desplegament

En el següent cronograma es marca de color blau (veure història), les tasques de cada iteració que han de passar per aquestes fases.

A més, el projecte es realitza a través d'un seguiment continu amb el client, això fa que cada x temps s'hagi de preparar una versió i tenir una reunió per desplegar-la a casa el client.

Com que les reunions per desplegar versió van d'acord amb els *sprints* de l'empresa, no s'inclouen en el cronograma, entenguen que cada tasca o grup de tasques es farà versionat i es desplegarà de cares al client. I que moltes tasques poden anar sortint amb el pas dels dies i les propostes per part de l'empresa o del client.

## 4.3 Cronograma temporal

### Història:

- En verd: Tasca/PR ok
- En verd: Test, versió i despliegament
- Fases: S'estimen a un temps determinat

ESTAT	ACTIVITAT	FASE 1		FASE 2		FASE 3		FASE 4		FASE 5		Juny
		Gener	Febri	Març	Abril	Mai	Juny					
Anàlisi de l'aplicatiu	Analisi de requeriments	1	2	3	4	1	2	3	4	5	1	1
	Models de treball										2	3
	Estructura esquemàtica del projecte										1	4
	Creació dels models del treball										5	5
Disseny de l'aplicació	Definició diagrames de casos d'ús, classe i seqüència											
	Estudi dels llibreries segons els requeriments de l'aplicatiu											
	Disseny de l'st de les llibreries											
	Disseny d'activitats i tasques											
	Definir projecte scrum											
Entorn de treball	Instal·lació de l'entorn de treball											
	Creació del repositori...											
	Definició de la llígera del software											
	Definició de l'estructura de classes del projecte											
	Estudi del entorn gràfic, modelats d'accés als recursos											
	Definir estructura de dades											
	Analisi de casos d'ús per un primer disseny d'interface d'usuari											
	Definició d'interface d'usuari											
	Definició de la comunicació											
	Estudi del llibreria per sockets											
	Definició i utilització de llibreria per comunicació amb escànders											
	Implementació de les primeres classes											
Desarrollament	Diseny de l'API											
	Implementació de la comunicació											
	Definir model E/R de dades											
	Estudi de llibreria per a recisió de la BD											
	Creació de la BD											
	Obrir document amb escaner											
	Obtenir detalls del document obtingut											
	Implementació de les preferències d'escaneig											
	Guardar a BD											
	Protocol base de Source Manager											
	Definir paths del projecte											
	Creació i desenvolupament base del sistema											
	Conversors de documents											
	Desenvolupament de l'interfície d'usuari											
	Desenvolupament d'un client de la vista											
	Desenvolupament de la comunicació											
	Desenvolupament de l'API											
	Definició del protocol Source Manager											
	Implementació del protocol Source Manager											
	Definició del plugin a ERP											
	Creació del plugin a ERP											
	Estudi de llibreria per executables instal·lador											
	Definició de les dependències per el setup											
	Creació del setup											
	Preparació de versió											
	Tests											
	Versió estable											

## 4.4 Fases de treball

(es presuposen iteracions setmanals aprox.)

### 1. Fase 1 Inici:

#### Iteracions: **IT1**

Requisits, idees, eines i *maquetatje* de l'aplicació: Inicialment només es compta amb la petició per part del client. Per tal calia preparar i definir un funcionament bàsic de l'aplicació, així com escollir les eines per fer-ho i fer un estudi dels avantatges i possibles inconvenients de la utilització d'una o altre.

En aquesta fase és important escollir unes bones eines per desenvolupar el software. Quan es tracta d'un projecte una mica gran, s'ha de tenir diverses coses en compte abans de començar:

Tasques:

- Adaptabilitat entre versions
- Suport
- Multi Arquitectura
- Anàlisi de requeriments
- Models de treball
- Estructura esquemàtica del projecte
- Creació dels models de treball

També en aquesta fase toca omplir píssarres i píssarres d'un petit esquema d'aplicació. Entre els mentors a l'empresa i jo vam fer com un petit *brainstorming* per començar a dibuixar i maquetar l'aplicació.

Cal definir una bona base de l'estructura esquemàtica del projecte abans de començar en temes de disseny. Cal tenir clar que es vol fer primer de tot abans de saber com es vol fer.

### 1. Fase 2 Diseny:

#### Iteracions: **IT2, IT3**

Definició i disseny del funcionament bàsic d'aplicació: Definir-ne el funcionament bàsic, així com les opcions, característiques, menús, accessos... del software. Básicamente emplenar més píssarres a mode d'esbós per començar a generar una idea del funcionament bàsic.

Utilització del llenguatge UML per definir esquemes i diagrames de disseny de l'aplicació. Esquemes de disseny:

- Disseny del diagrama de casos d'ús de context
- Disseny del diagrama de classes principal: Es defineix el diagrama de classes principal així com els seus mètodes
- Disseny de diagrames de seqüència: Diagrames de seqüència dels principals casos d'ús, com són:
  - Obtenir document
  - Adjuntar document
  - Crear preferències

També es defineixen les estructures bàsiques del projecte, i es fa un replantejament de les llibreries que millor compleixin els requisits i les funcionalitats a desenvolupar, i es realitza un estudi exhaustiu de les mateixes tal com una utilització bàsica per fer una extracció d'avantatges i limitacions.

Tasques a realitzar:

- Definició diagrames de casos d'ús, classe i seqüència
- Estudi de llibreries segons els requeriments de l'aplicació
- Disseny de l'ús de les llibreries
- Disseny d'activitats i tasques
- Definir projecte scrum

### 1. Fase 3 Entorn de treball:

Iteracions: **IT3**

Es compta que l'ERP està instal·lat i funcionant, però cal configurar el marc de treball necessari per desenvolupar, testejar i compilar tant la part d'aplicació d'escriptori, com la part web.

Inicialment es decideix la utilització de certes llibreries i entorns de treball, però les necessitats que requereixi l'avançament en el desenvolupament del software farà que se'n necessitin i n'apareguin de noves. En capítols posteriors es definirà l'entorn de treball.

Tasques a realitzar:

- Instal·lació de l'entorn de treball

### 1. Fase 4 Creació/Start:

Iteracions: **IT4, IT5, IT6**

En aquesta fase es crea el projecte, amb una estructura bàsica definida en fases anteriors. És important tenir escollida la metodologia de desenvolupament, ja que es tracta d'un projecte de software i és interessant utilitzar sistemes que estan a la nostra disposició com git.

Aspectes a tenir en compte:

Repositori: Es crea un repositori a GitHub per seguir el treball.

Implementació de classes: Seguint l'estructura definida es maqueten les classes principals amb funcionalitats molt bàsiques per utilitzar un desenvolupament escalar tot fent cas del mètode “divideix i venç”

Comunicació: Per comunicar els diferents mòduls de l'aplicació s'utilitzen crides IPC a través de *sockets*.

Entorn gràfic: Esbós per maquetar la vista d'usuari.

Tasques:

- Creació del repositori
- Definició de la llicència de software
- Definició de l'estructura de classes del projecte
- Estudi d'entorn gràfic i modelats d'accés als recursos
- Definir estructura de dades
- Anàlisi de casos d'ús per un primer disseny d'interfície d'usuari
- Definició d'interfície d'usuari
- Definició de la comunicació
- Estudi de llibreria per sockets
- Definició i utilització de llibreria per comunicació amb escànners
- Implementació de les primeres classes

### 1. Fase 5 Desenvolupament:

Iteracions: de la **IT7** a la **IT18**

Aquesta fase inclou totes les tasques de desenvolupament per crear una versió que compleixi els requisits funcionals plantejats. En aquesta fase és molt important tornar-se metòdic i organitzar molt bé les tasques. Es proposa:

- Implementar poques o fins i tot una funcionalitat alhora:
  - Per cada funcionalitat crear: Issue, Pull Request, Branca i tag
- Durant la implementació centrar-se en deixar llista la funcionalitat
- Tests: un cop “acabada” es realitzaran testos exhaustius amb un patró de tests, sempre seguint els mateixos i comprovant testos anteriors. És a dir comprovar que l'agregació d'aquesta nova funcionalitat no interferèixi o canviï el comportament de les que s'havien realitzat fins ara.

La fase de desenvolupament va molt lligada a l'ús de certes llibreries, les quals és necessari conèixer una sèrie de característiques, que prèviament s'hauran pogut estudiar o analitzar. A l'avanc del programa, aniran sortint diverses necessitats, i d'aquestes es necessitaran cobrir, moltes amb certs paquets. És per això que aquest punt detalla la planificació inicial, però que obviament no serà exactament fidel en tot el

desenvolupament, ja que sempre poden sortir coses noves, i objectivament és bo per l'aplicació.

Algunes de les iteracions més importants d'aquesta fase inclouen:

- Connexió d'un escàner de proves, obtenir accés amb una certa llibreria que es presentarà més endavant, i operacions amb la llibreria
- Scanner-Plugin: Implementació d'un plugin a l'ERP pel Gestor documental.
- Reestructuració d'aplicació: Els requeriments del software i la idea i ganes de desplegar una arquitectura modelable fan que es replantegin alguns elements del disseny de l'aplicació, per tant es dedica aquesta fase a afinar certs mòduls, crear-ne de nous, definir-los i separar-los bé utilitzant mètodes d'herència...
- Base de dades: disseny i implementació de la base de dades per guardar els perfils i preferències de digitalització.
- Drivers: Treball amb controladors d'escàners i versions entre 32 bits i 64 bits
- VPN: Obrir un enllaç cap a la VPN de l'empresa que sol·licita el producte, per tal d'obtenir accés a un escàner seu per realitzar proves.
- Realització de proves amb el software que es desenvolupa amb un cert escàner, testeig de drivers, compatibilitats amb la llibreria d'accés als escàners.
- Desplegar primera versió a preproducció: Desplegar la primera versió del software a preproducció, demo, tests i comprovacions amb el client. Discussió de possibles millors, detecció d'errors i correcció d'aquests últims

Tasques:

- Disseny de l'API
- Implementació de la comunicació
- Definir model E/R de dades
- Estudi de llibreria per la creació de la base de dades
- Creació de la base de dades
- Obtenir document amb escàner
- Obtenir detalls del document obtingut
- Implementació de les preferències d'escaneig
- Guardar a la base de dades
- Protocol base de Source Manager
- Definir paths del projecte
- Creació i desenvolupament base del sistema
- Conversors de documents
- Desenvolupament de l'interfície d'usuari

- Desenvolupament d'un client de la vista
- Desenvolupament de la comunicació
- Desenvolupament de l'API
- Definició del protocol Source Manager
- Implementació del protocol Source Manager
- Definició del plugin a ERP
- Creació del plugin a l'ERP

1. Fase 6:

Iteracions: de la **IT19, IT20, IT21**

Per últim cal generar una versió final i estable del software per poder-la desplegar a casa al client. Òbviament durant el procés del projecte s'aniran creant i desplegant versions, però com a fi de TFG es generarà una versió final del producte, tot i que un cop acabat es continuaran fent versions per al client i millorar el producte.

Tasques:

- Estudi de llibreria per executable i instal·lador
- Definició de les dependències pel
- Creació del setup
- Preparació de versió
- Tests
- Versió estable

## 4.5 Valoració de la planificació

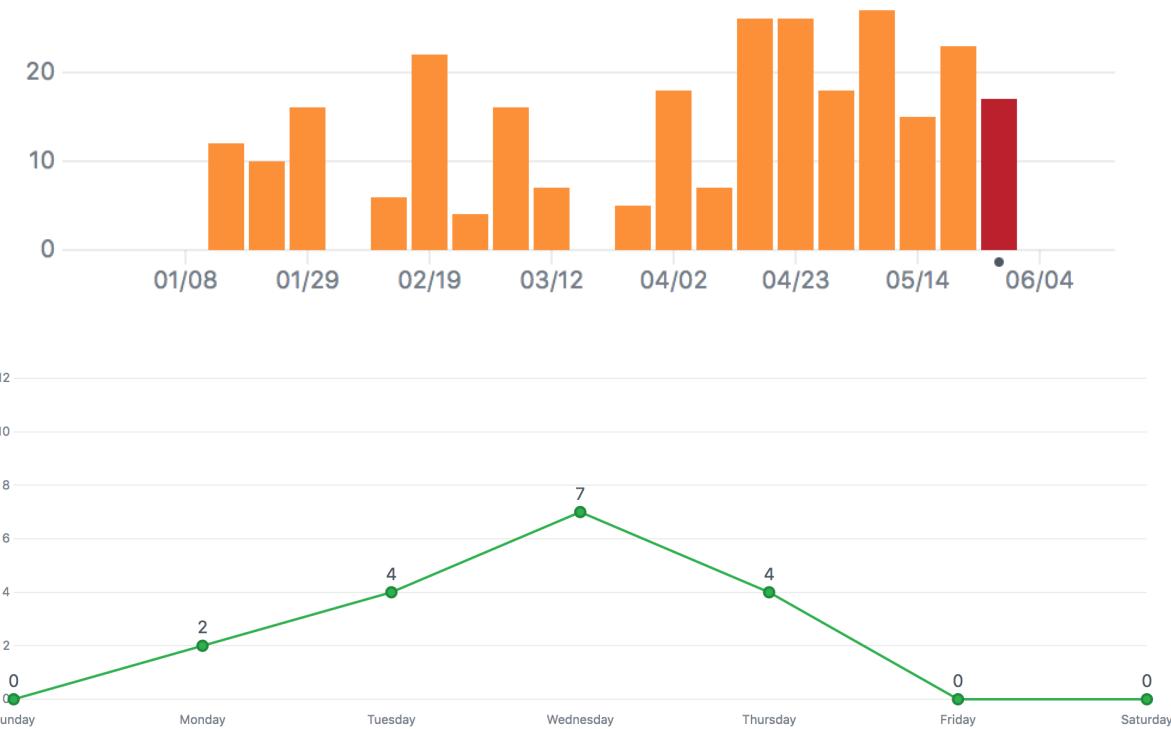
Les tasques s'han anat complint d'acord amb la planificació excepte algunes que s'han allargat fins a la iteració següent. Això és normal degut que quan es fa una planificació s'ha de ser optimista però s'ha de tenir en compte que el marge de decisió és molt gran. És a dir, inicialment quan es planteja una tasca es pressuposa que no hi haurà limitacions o traves que provoquin un endarreriment, doncs bé, això passa, i és ben real. Ja sigui per motius de limitació d'una llibreria, o per motius de requeriments no previstos... A més cal comptar que aquest software es desenvolupa sota demanda del client, i que durant el desenvolupament, s'han demanat moltes funcionalitats noves que en un principi no es contemplaven.

Per ocupació de l'aplicació principal, ha quedat fora de l'abast d'aquest treball el desplegament web. S'ha començat amb React.js i s'ha iniciat una interfície bàsica que es mostra a l'apartat de resultats. En un treball futur es continuará amb el desenvolupament de la web.

## 4.6 Freqüència de desenvolupament

Tal com es marca en la planificació, les fases d'iteracions han estat constants i regulars. Si s'observa el següent gràfic es pot apreciar que els tancaments dels sprints són els dijous, per tant els dimecres cal deixar tancades totes les tasques que sigui possibles (veure càrrega de treball), així no s'arroseguen d'una iteració a l'altra.

Gràfica de la càrrega i freqüència de desenvolupament:



És evident que la planificació del temps estimat d'una tasca no sempre concordarà en el temps real de realització, però és important intentar cenyir-se al màxim a les iteracions per treure suc a aquest tipus de metodologia. Utilitzant el sentit comú i les bones pràctiques, es torna un mètode eficient.

# CAPÍTOL 5

---

## Marc de treball i conceptes previs

---

Abans de posar-se a desenvolupar directament, s'ha de fer un estudi previ i obtenir els coneixements necessaris per desenvolupar en un seguit de llibreries, *frameworks*, motors gràfics...

Amb la intenció de situar al lector en el marc actual de treball, a continuació es descriuen les característiques generals del sistema:

- El sistema treballarà independentment
- Comunicació paral·lela amb l'ERP del client
- Sistemes Windows: s'haurà de desenvolupar una versió per windows, ja que les màquines del client que ha sollicitat el software treballen amb aquest sistema. En aquest treball es dissenya i es defineix un sistema multiplataforma, però l'abast del desenvolupament queda centrat en sistemes windows. En capítols posteriors es detalla millor el perquè i el com.

En termes d'aquest TFG, per fer una breu introducció a l'estruatura, a continuació es comparteixen els detalls necessaris per entendre l'estruatura inicial, el funcionament bàsic, i l'estruatura a desenvolupar.

El principal és crear un software que es comuniui amb un ERP per tal de transferir-hi els documents acabats de digitalitzar i així automatitzar el procés de classificació. Definides les característiques de treball, fóra bo senyalar com funciona i com està desplegada cada característica per tal de situar al lector en un marc més familiar.

### 5.1 ERP

Basat en OpenERP, i atacat generalment en Python, actualment Python 2.7. Com que en aquest projecte es treballarà de manera paral·lela en aquest ERP, inclòs es desenvoluparà algun mòdul per tal d'integrar-hi el software de digitalització i gestió documental, val la pena fer-ne un petit esment. Aquest ERP està desplegat en els servidors de producció de cada client, i per tant es treballa amb l'ERP instal·lat físicament a cada màquina, sinó que està en un entorn distribuït. Això és un gran avantatge a l'hora de desplegar canvis,

possibles fallades d'una màquina física en concret... Els empleats es connecten i treballen amb aquest ERP. El mètode de treball i desenvolupament sobre aquest ERP és interessant, ja que se segueixen unes directrius molt marcades a l'hora de preparar versió, realitzar canvis amb un sistema organitzat de branques amb developer, integració... però l'únic que interessa en aquest treball és saber que per desenvolupar el mòdul necessari pel software caldrà seguir aquestes pautes correctament. Per desenvolupar en aquest entorn cal tenir accés al repositori, d'aquí es descarregarà el codi, i es desenvoluparà el necessari directament amb un IDE de programació. Cal senyalar que per treballar correctament es crearà una branca de treball per poder aplicar els canvis de manera sistemàtica i còmode.

**Nota:** Com a concepte previ, remarcar que ja existeix una versió de l'ERP per sistemes Windows. És una versió client (OpenERPClient) que es connecta al servidor on està desplegat l'ERP, i que per instal·lar-lo hi ha preparada una versió .exe per executar-lo. Un cop desenvolupada la part de l'ERP es pot utilitzar un fitxer setup.py per generar un nou executable per afegir els canvis a la nova versió.

## 5.2 Base de Dades

Cal tractar encara que sigui breument aquest tema, ja que quan es treballa en documents, una de les coses més importants és la persistència en disc i la consistència de tals. A l'empresa, per gestionar l'ERP, s'utilitzen dos motors de base de dades: *PSQL* i *MongoDB*. El primer s'utilitza per a tot el tractament de les dades, i el segon en ser un sistema no relacional és molt avantatjós per la persistència d'arxius en disc.

## 5.3 Entorn bàsic de desenvolupament

GISCE es preocupa pel software lliure, tots els desenvolupadors utilitzen un S.O. software lliure, i la majoria d'eines per no dir totes també segueixen les mateixes directrius. Però el software que es desenvoluparà durant aquest TFG haurà de córrer en sistemes Windows, ja que els usuaris que utilitzaran aquest software i l'ERP, utilitzen aquest tipus de sistemes. Òbviament es podria desenvolupar amb el sistema que es vulgui i compilar-lo posteriorment per Windows, però per interactuar amb els escàners, es farà servir una llibreria feta per Windows. Com que ha de desplegar-se en Windows, cal si o si una llibreria que funcioni en aquest sistema. Per tant es desenvoluparà en aquesta plataforma. Quan es toquin mòduls de l'ERP o es desenvolupi el visor web (si hi accompanya la temporització), s'utilitzaran probablement, sistemes Unix. Apuntar que s'utilitzarà una màquina virtual muntada sobre un sistema de virtualització, i anotar que Microsoft Windows posa a disposició una sèrie de sistemes virtualitzats per desenvolupadors, testers o simplement usuaris que necessitin un sistema Windows de curta durada sense haver d'obtenir-ne la llicència. Aquests sistemes són totalment legals i no violen cap dret de la propietat intel·lectual.

<https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/>

Fent una visió general, el *material* necessari per desenvolupar és: Un IDE de programació, intèrpret de Python, un entorn virtual... Vegem-ne la llista:

- IDE: PyCharm (llicència educativa / Community)
- Intèrpret de Python: *Python 3.6* i *Python 2.5* en una versió antiga de l'ERP Client
- Git
- Windows Virtualitzat: (Virtualbox, testejant amb versions x86 i x64)

- OpenVPN: Per realitzar proves amb l'escàner del client

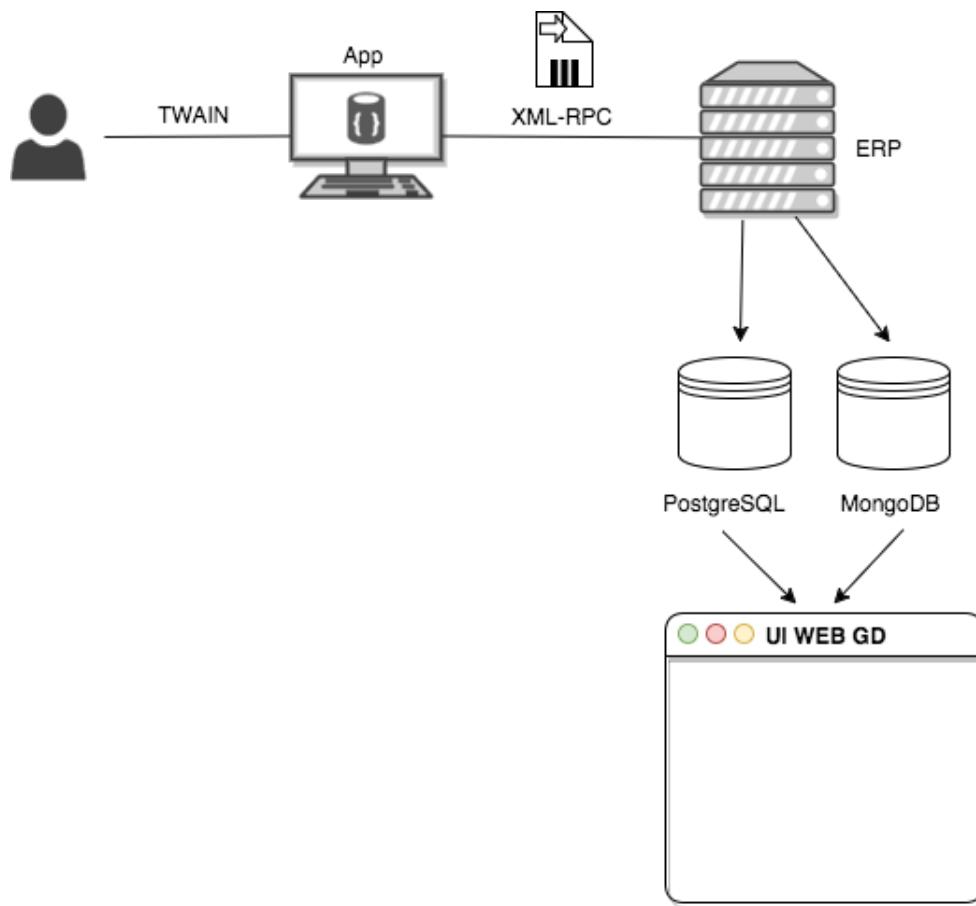
## **5.4 Sistema a desenvolupar**

Per entrar més en matèria i tenir una idea més clara del projecte a desenvolupar i les respectives tasques, a continuació se'n fa una descripció:

La idea més bàsica sense definir els detalls de les funcionalitats i extres del sistema, és desenvolupar un software d'escriptori que permeti:

1. Digitalitzar documents amb unes certes característiques
  - 1.1. Format de sortida
  - 1.2. Color
  - 1.3. Resolució
  - 1.4. Multipàgina
  - 1.5. ...
2. *Validations:* Comprovació de documents vàlids
3. Adjuntar i classificar de manera automàtica cap al software de l'altre extrem (en aquest cas l'ERP)
  - 3.1. Qualsevol software és vàlid, ja que el software és adaptable. L'àbast d'aquest treball se centra en un ERP
4. Gestió del document via ERP
5. Gestió del document via Web
  - 5.1 Consulta dels documents
  - 5.2 Visualització dels documents

A continuació es detalla una gràfica bàsica amb procés de treball del sistema:



Com mostra l'anterior figura, l'usuari gestionarà l'aplicació des del seu escriptori per digitalitzar els documents necessaris, un cop acabada la tasca, el software adjuntarà de manera automàtica i intel·ligent els documents on correspongui de l'ERP.

Per tenir una idea més amplia de la seva utilització, es detalla a mode llista el treball de l'usuari:

1. Usuari està amb l'ERP i vol escanejar
2. Utilitza un botó que obre el programa per digitalitzar
3. Digitalitza el/els documents
4. Revisa, edita, previsualitza els documents
5. Confirma que ha acabat de treballar amb el programa
6. El programa es tanca i adjunta automàticament els documents

La intel·ligència del software fa que els documents s'adjuntin de manera classificada.

## 5.5 Marc de treball per desenvolupar un programa d'escriptori

En la primera i les següents fases de desenvolupament d'aquest TFG es defineixen els motors, eines, llibreries... necessaris per desenvolupar-ne el treball. Però és evident que l'avanc del mateix pot necessitar

readaptar-se o se'n poden anar afegint característiques que requereixin l'ús de noves eines. Per definició el software d'escriptori que es vol desplegar necessita de:

- Una vista d'usuari
- Una base de dades per guardar preferències i perfils de digitalització
- Comunicar-se amb altres processos
- Utilitzar-se paral·lelament des d'un altre software
- Obtenir accés als escàners (Twain en Windows i Sane en Unix)

### 5.5.1 Vista d'usuari

Es presenta a l'usuari un entorn amigable de finestres per tal d'interactuar amb l'aplicació. PyQt ha estat l'eina escollida per desenvolupar aquest sistema.

### 5.5.2 Base de Dades de Preferències i perfils de digitalització

Permet a l'usuari predefinir unes preferències de digitalització. És a dir, permet escollir les preferències en què es volen digitalitzar els documents (color, resolució, doble cara, on guardar el document...). A més una ampliació que presenta l'aplicació és que molts, per no dir tots els drivers dels escàners no permeten escollir el format de sortida (pdf, png, bmp...). Aquesta aplicació permetrà escollir aquesta preferència. Les preferències bàsiques seleccionades es podran guardar i utilitzar sempre que es vulgui. Fent així molt més ràpid el seu ús. El sistema guarda aquestes preferències perquè estiguin sempre disponibles. A més permetrà definir un sistema de perfils, per guardar i tenir diversos perfils amb diferents preferències cadascun.

### 5.5.3 Comunicar-se entre processos

Previ als temes següents cal senyalar que l'aplicació d'escriptori, en el marc d'aquest treball no *correrà* sola. És dir serà una aplicació que treballarà de la mà de l'ERP mencionat anteriorment. És important senyalar que és una aplicació apart d'aquest ERP, però que si caldrà comunicar-s'hi, per tal d'afegir una intel·ligència a l'hora d'adjuntar documents. El sistema serà iniciat des de l'ERP. És a dir, l'usuari estarà treballant amb aquest, i quan decideixi digitalitzar un document, sols haurà de navegar amb l'ERP a on necessiti (pòlissa d'un client, empreses, facturació...), fer clic a un botó i directament passarà a treballar amb aquest software.

### 5.5.4 Obtenir accés als escàners

Com a conceptes tècnics, per tal de treballar amb els escàners, s'utilitzarà la llibreria twain, que es detalla en capítols posteriors.

## 5.6 Marc de treball per desplegar una aplicació web

Es desplegarà una petita aplicació web per tal de navegar amb els documents obtinguts a través del software d'escriptori que s'ha desenvolupat.

Aquest apartat és un objectiu del treball, i si la temporització i la feina que doni l'apartat principal no hi accompanyen, quedarà fora de l'abast d'aquest TFG.

La idea és un navegador de documents, per poder navegar entre els objectes i documents adjunts persistents a la base de dades de l'ERP.

## **5.7 Nom comercial ScannerApp**

Com a nom comercial pel software a desenvolupar s'ha escollit *ScannerApp*, i serà freqüent a partir d'ara, referir-se a l'aplicació amb aquest nom.

# CAPÍTOL 6

---

## Requisits del sistema

---

Aquest software ha de permetre a la digitalització, organització, visualització... de documents. Entrant en detall de les funcionalitats i no funcionalitats dels requisits del sistema, a continuació se'n llisten les característiques:

### 6.1 Iniciar el software des de l'ERP

ScannerApp permet treballar independentment o en paral·lel amb un altre software. En el marc d'aquest TFG treballarà directament amb l'ERP de l'empresa. Tot i que corren dos processos separats, cal mencionar-ne els detalls: Ha de permetre a l'usuari iniciar el software a través d'un botó de l'ERP. Per més claredat s'exposa el funcionament: L'usuari estarà treballant amb l'ERP, quan vulgui digitalitzar un document, haurà de seleccionar el botó "Plugins" - "Scan". A continuació s'iniciarà el programa, on podrà treballar amb les funcionalitats que s'exposen a continuació

### 6.2 Classificació de documents

És important senyalar, que el software *classificarà i adjuntarà els documents* al recurs de l'ERP des d'on s'-hagi iniciat l'ScannerApp. Els recursos de l'ERP són els apartats i objectes típics amb els quals treballen les empreses elèctriques. La pòlissa d'un client, empresa, facturació, CUPS, etc. Per exemple: Si l'usuari està treballant amb la pòlissa d'un client, i inicia l'ScannerApp, tots els documents que obtingui, s'adjuntaran automàticament en aquesta pòlissa.

Aquest procés automàtic de classificació és molt ràpid i aporta molts avantatges, ja que els clients estan acostumats a treballar des de l'ERP a *mode llista*, que és quan van recorrent recurs a recurs de l'ERP fent les tasques necessàries, si s'han d'adjuntar diversos documents, l'ScannerApp és molt còmode perquè permet abstreure al client d'aquesta classificació, anant recurs a recurs en mode llista i digitalitzant la pila de documents que pugui tenir. Potser el gran avantatge és quan el client està treballant amb l'ERP, i arriben diversos

clients alhora a entregar documents (molt freqüent a l'empresa on han demanat el software, de fet un dels grans motius pel qual l'han demanat és aquest), el client treballant amb l'ERP només s'haurà de preocupar de navegar fins al recurs corresponent de l'ERP, i al treballar amb l'ScannerApp, s'abstreu totalment de: si ha barrejat documents, o no sap on els ha desat, o si els adjunta i després els vol previsualitzar, o si ha de repetir el document i després no sap quin és el bo, o si ha d'anar a la carpeta a eliminar-lo, o si ha de canviar el nom del document, o de si n'ha adjuntat uns quants i ha perdut el recurs on treballava, o de si cada cop ha de seleccionar el mateix escàner i les preferències de digitalització, o que cada controlador de l'escàner sigui diferent, o si... I de tot això no se n'ha de preocupar perquè els documents d'aquella sessió es classifiquen al recurs actual, i la feina amb l'ScannerApp haurà estat fàcil, ràpida i metòdica.

### 6.3 Escàners disponibles

Aquesta característica informa a l'usuari dels escàners disponibles. Per utilitzar aquesta funcionalitat, es proporciona un botó, per tal de comprovar quins orígens estan instal·lats i disponibles. Aquesta característica permet tenir coneixement de si l'origen està ben instal·lat i funcional, si els controladors estan en funcionament, etc. **nota:** orígens, es refereix a un dispositiu de digitalització de documents (escàner, càmera...)

### 6.4 Preferències

És molt freqüent voler o haver de digitalitzar un document amb una sèrie de característiques, color, resolució... L'aplicació ha de permetre a l'usuari escollir unes preferències. Per fer-ho disposarà d'un botó que llançarà una finestra amb un selector de preferències a escollir. Entre aquests:

- Escàner: es proporciona una llista d'escàners disponibles, en seleccionar-ne un, és guarda com a escàner per defecte
- Color: permet escollir entre color, escala de grisos o blanc i negre
- Dpi: permet escollir la resolució del document a digitalitzar
- Format de sortida: permet escollir entre bmp, pdf o tiff
- Eliminar documents al sortir: marcant aquesta opció, els documents no persisteixen en disc

### 6.5 Digitalització Nativa

L'usuari tindrà l'opció de digitalitzar els documents utilitzant el controlador de l'escàner. Per detallar-ho més aquest és el procés:

- Usuari selecciona digitalització nativa: l'aplicació disposa d'un botó per fer-ho
- Seleccionar origen: Si l'usuari no ha predefinit cap escàner, se li proporciona una finestra per escollir quin escàner vol utilitzar en aquest procés de digitalització
- Controladors natius: A continuació se li presenta a l'usuari una finestra proporcionada pels controladors instal·lats al sistema, on pot escollir els paràmetres que el fabricant proporciona

- Previsualització: l'usuari pot previsualitzar el document abans d'iniciar la digitalització per estar segur de que és el que vol i està com vol

Aquest procés és interessant per adaptar-se totalment a les necessitats de l'usuari, així pot digitalitzar d'una forma coneguda i còmode pel seu ús de treball.

## **6.6 Digitalització**

Una de les tasques més tedioses quan s'ha de realitzar un procés de digitalització repetitiu, és seleccionar cada cop el mateix origen (escàner), configurar-ne les preferències, inclús depèn del fabricant, primer obliga a previsualitzar el document i després digitalitzar-lo, tot aquest temps es fa molt llarg si es té clar el que es vol, es podria dir que és sobrant. Aquesta opció permet a l'usuari amb el sol clic d'un botó, digitalitzar un document, sense preocupar-se de res més. Això és gràcies al fet que lleixa les preferències que ha definit l'usuari, i són persistents a la base de dades. Per tant en seleccionar aquesta opció l'usuari no s'ha de preocupar absolutament de res.

## **6.7 Perfiles**

L'aplicació permet la creació de perfils, per tal que l'usuari pugui establir diferents perfils de digitalització, i utilitzar-ne els que vulgui en cada moment.

## **6.8 Edició i previsualització**

L'usuari pot visualitzar i editar les característiques del document obtingut segons uns formats típics, com els que es detallen a continuació:

### **6.8.1 PDF**

Es disposa d'un previsualitzador enriquit de documents PDF, on entre d'altres podrà navegar entre pàgines (si és el cas), canviar el nom del document, obrir-lo des del sistema d'arxius... També permet obtenir informació del document com la mida, la resolució, l'hora d'obtenció...

### **6.8.2 BMP**

Permet una vista prèvia de la imatge, així com el canvi de nom del document, i obrir-lo des del sistema d'arxius. També permet obtenir informació del document com la mida, la resolució, l'hora d'obtenció...

### **6.8.3 TIFF**

Permet una vista prèvia de la imatge, així com el canvi de nom del document, i obrir el document des del sistema d'arxius. També permet obtenir informació del document com la mida, la resolució, l'hora d'obtenció...

Aquesta funcionalitat es maneja a través d'un sistema de navegació de pestanyes, que es defineix més àmpliament al següent apartat.

### 6.9 Tractament i selecció

L'aplicació inclou una vista principal que conté una llista interactiva amb els arxius obtinguts. Per cada document se'n mostra el següent:

- Nom i ruta completa: El nom i la ruta absoluta del document.
- Mida: La mida de l'arxiu.
- Hora: Hora en precisió dia, mes, any, hora, minut i segons de la digitalització del document.
- Format: Extensió del document.
- Edició: Existeixen dues maneres per editar i previsualitzar un document:
  - *clickable*: En fer dos clics sobre el document s'entrarà al mode seleccionat
  - Botó edició: En seleccionar un document, s'activa un botó per entrar al mode edició
- Eliminació: Cada document té un botó extra per eliminar-lo.

**nota:** en el repositori del projecte s'hi deixa un fitxer de requeriments per desplegar l'entorn de desenvolupament en qualsevol màquina que disposi d'un intèrpret de Python.

### 6.10 Systray Icon

Afegeix una icona de control del programa al systray. En capítols posteriors es tracten temes de disseny intern de l'aplicació, però a tall de resum, cal dir que l'aplicació consta de dos processos, un de “pare” que posa en marxa el “fill”. El pare no té interfície gràfica i per donar un feedback i agregar funcionalitats pràctiques al programa, permet un control des de la icona de systray. Imaginem que el procés “pare” està en funcionament, però no ha sol·licitat l'enengada del procés “fill”, si no tenim interfície gràfica és un punt incòmode haver de consultar els processos del sistema per saber si el programa està corrent, en canvi amb el systray icon, no cal fer res, la icona a la barra systray ja ens indica que està corrent.

Modes de selecció:

- Obrir els logs
- Ocultar icona
- Tancar programa

**nota:** totes les icones de l'ScannerApp són Public Domain, extretes del lloc web: <https://thenounproject.com/>, pel qual no tenen llicències Creative Commons i no cal citar-ne l'autor.

## **6.11 Requisits no funcionals**

### **6.11.1 Emmagatzemament i ús**

El software està dissenyat per tal de poder funcionar en una màquina o servidor. El sistema està pensat per poder executar-se en diferents arquitectures, però queda fora de l'abast d'aquest treball. Actualment està treballant sobre sistemes Microsoft Windows. Tot i que es pot executar en sistemes diferents a Windows, el funcionament no seria el correcte, ja que una de les llibreries principals que utilitza és per Windows.

Cal disposar d'una màquina amb un espai lliure mínim de 70mb aprox. per realitzar la instal·lació, i un espai addicional per si utilitzem la funció de guardar els documents al sortir (com es detalla en capítols anteriors, una de les funcionalitats del sistema és escollir la persistència dels documents a disc).

### **6.11.2 Fiabilitat**

Sens dubte una tasca important a l'hora de desenvolupar un sistema o software és tenir en compte la fiabilitat, obviament és una gran responsabilitat del programador, tot i que a vegades hi ha factors que es poden escapar a la previsió inicial o que simplement surten del marc o de l'entorn de treball habitual del sistema.

Aquest software s'ha desenvolupat tenint molt en compte aquest aspecte, principalment perquè a través del seguiment client-desenvolupador que s'ha anat seguint durant el procés, a l'hora de desplegar les diferents versions d'aquest software, una de les coses més importants a què no tingui fallades és la fiabilitat. Amb això es pretén explicar que a l'hora de fer les reunions de cada *sprint* que entrava una versió nova d'aquest software, era important que fos fiable, per diversos motius, però és clar també perquè no fallés directament a la màquina del client.

Tot sovint no és trivial detectar certs errors, és clar que amb unes bones pràctiques i debugant es detecten i se solucionen uns amb més facilitat que d'altres, però aquest punt vol tractar errors més difícils de detectar, més complicats d'aparèixer i més atípics.

El software incorpora dos grans sistemes de controls d'errors i monitoratge:

#### **Sentry**

Sentry és un sistema de monitoratge i captura d'events i excepcions. Permet capturar events de l'aplicació i llençar-los al sistema sentry per poder detectar errors. Sentry té diversos nivells de captura d'errors, però és interessant saber que és una opció molt bona de cara al monitoratge i captura d'errors, ja que envia al sistema sentry tot el traceback d'error, el moment just, el servidor o màquina on s'ha produït la fallada, quin era l'estat de l'aplicació i les dades en el moment de la fallada.

Genera un informe complet de tot el que passa i el que ha passat. En la següent figura es mostra el sentry integrat en el projecte:

The screenshot shows the Sentry interface for the 'ScannerApp' project. On the left, a list of errors is displayed with their counts (e.g., 27, 1, 2, 10, 24, 6, 2) and descriptions. On the right, various filters are available to refine the search results.

Com ja s'ha anat comentant, el projecte avança paral·lelament amb els esprints de l'empresa que ha demanat el software. Al desplegar l'aplicació en aquesta empresa, sentry pot ajudar molt a monitoritzar les fallades i saber que passa a la màquina o servidor del client. En aquest cas l'empresa que ha sol·licitat el producte i que actualment ja té desplegada una de les primeres versions del projecte es troba a Andalusia, obviament Sentry hi juga un paper més clau, ja que ens acosta directament al que està passant. La següent figura mostra de manera detallada la captura d'una fallada en el software desplegat allà. Sense haver-nos de moure, sabem exactament què ha fallat, en quin moment, i a on.

This screenshot provides a detailed view of a single error captured by Sentry. It includes the event ID, timestamp, and a link to the full event data. The error itself is a 'TypeError' occurring in the 'scannerapp/agent/connector.py' file at line 131. The stack trace shows the call chain from 'recvMsg' to 'send'. On the right, there are options to 'Mute Event' or 'Remove Event Data', along with specific details about the error context like the level ('error'), logger ('root'), and server name ('ucecajbp').

L'adreça de monitoratge del Sentry és: <http://sentry.gisce.net/devel/scannerapp/>

## **Logger**

El sistema anota informacions i fallades en fitxers .log per tal de tenir una noció més completa del que passa. Els fitxers log es poden consultar navegant fins al directori per defecte. Aquests fitxers es guarden per defecte al “home” de l'usuari. Aquesta ruta es pot canviar si es prefereix, però per temes de permisos s'utilitza aquesta en fases de desenvolupament.

El logger consta de dues parts:

- Logger de Vista
- Logger de Servidor

En la ruta predefinida es troben els dos logs amb nom per defecte: logs\_gui, server\_logs. Els dos informen cada cop que s'inicia el software, qualsevol fallada, obertura de sockets... En registra d'on prové i l'hora de la captura.

**nota:** Ruta d'accés als logs: %userprofile% ó \$home



## Estudis i decisions

---

### 7.1 Requisits segons el sistema de l'empresa

El desenvolupament d'aquest software es realitza a l'empresa, i per tant està integrat en aquesta. L'aplicació ha de poder-se iniciar des de l'ERP, tot i que és un software apart, ha de permetre integrar-se de manera correcta. El desenvolupament va de la mà dels requisits del sistema de l'empresa. Per desenvolupar i fer que sigui un sistema funcional cal el següent:

- Escàners
- Sistema Operatiu Windows
- ERP de l'empresa: Aquest software és propietat de l'empresa GISCE-TI, S.L. i aquest TFG pretén integrar el software que es desenvolupa a mode *plugin* en aquest ERP.

Els punts esmentats són els requisits mínims de l'entorn de treball i defineixen l'estructura bàsica de treball, sense fer esment a les eines, llibreries... que s'utilitzen per desenvolupar el software. A continuació se'n detallen les bàsiques, i el perquè s'han escollit aquestes.

És molt important escollir unes bones eines de desenvolupament. Subjectivament sóc partidari d'invertir més temps en una bona preparació, definició i configuració de l'entorn de treball, per la qual s'acaba recuperant el temps invertit, ja que el treball es torna metòdic i àgil.

El llenguatge de programació principal per al gestor documental s'ha escollit Python. Concretament la versió Python 3.6.

### 7.2 El perquè de Python

- Fàcil utilització
- Llenguatge interpretat

- Empresa consagrada en aquest llenguatge
- Suport
- Bon manteniment
- Moltes llibreries
- Software Lliure

### 7.3 Accés a Escàners

La llibreria escollida per accedir a un escàner, i poder treballar amb ell a través de la programació és: TWAIN. De fet, twain és un estàndard definit pel grup twain.org, destinat a l'adquisició d'arxius des d'una font digital com és un escàner, càmera... Una API de captura d'imatges per Sistemes Operatius Windows i Apple Macintosh.

Per la utilització i accés d'aquests estàndards amb Python, existeix una llibreria coneguda com a pytwain. Per instal·lar pytwain, es pot fer mitjançant PyPi, pip, easy\_install... Objectivament l'eina més utilitzada és pip.

Pip és un gestor de paquets, que a més de les funcionalitats bàsiques permet actualitzacions, cerques, desinstal·lacions, instal·lacions des d'un fitxer de requeriments, i és capaç de mostrar l'informació de les versions específiques de cada paquet instal·lat en el sistema. Pip està inclòs en la majoria de versions de l'interpret de Python, excepte si s'utilitzen versions molt antigues i pràcticament obsoletes. Llavors s'ha d'obtenir pip mitjançant un mètode anomenat: get-pip.

Per tenir una idea bàsica de funcionament de Pip, es posa com a exemple la instal·lació de la llibreria pytwain.

Per buscar paquets:

```
pip search twain
```

Per instal·lar la llibreria:

```
pip install pytwain
```

Per mostrar les versions de llibreries instal·lades al sistema:

```
pip freeze
```

o

```
pip list
```

Per instal·lar versions concretes:

```
pip install pytwain==2.0.1
```

```
pip install pytwain>2
```

```
pip install pytwain<2
```

Per instal·lar llibreries des d'un fitxer requirements.txt:

Python permet crear un fitxer pla amb els requeriments que el software necessita per treballar. Això és molt útil per desplegar el software en diversos sistemes, a l'hora de fer-ho Pip dóna la facilitat d'instal·lar automàticament tots els requeriments esmentats en el fitxer de requeriments. Per fer-ho cal crear un fitxer pla amb el nom dels mòduls a instal·lar, i opcionalment la versió de cada un. Si no s'especifica cap versió, automàticament pip instal·larà la més nova. Per utilitzar un fitxer de requeriments només cal:

```
pip install -r requirements.txt
```

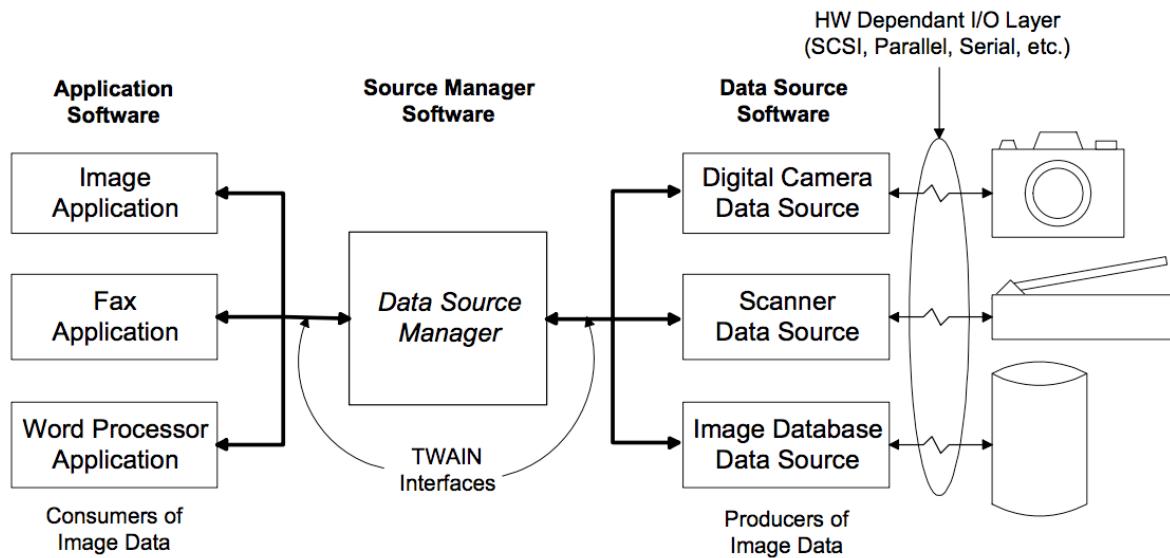
Twain permet accedir i treballar directament amb el que anomenen Data Source. Pot treballar amb càmeres, escàners... En aquest TFG s'utilitzen els Data Source de tipus escàner.

El que proposa aquesta llibreria és una graph d'estats, els quals cadascun defineix un estat de l'aplicació i un tipus d'accés. Twain proposa un “camí” per saltar d'un estat a un altre. Per definir-ho d'una manera més trivial, ens imaginem que hem d'accedir a un escàner i digitalitzar. Posem a tall d'exemple que accedir a un escàner és l'estat 1, i digitalitzar és l'estat 2. Twain proposa: que per accedir a l'estat 2 s'ha de passar per l'estat 1, no es pot accedir a l'estat 2 estant en cap altre estat.

A continuació es detalla com funciona twain, i més endavant es detallaran alguns aspectes bàsics de com ha ajudat aquesta llibreria en el projecte, i com s'ha utilitzat.

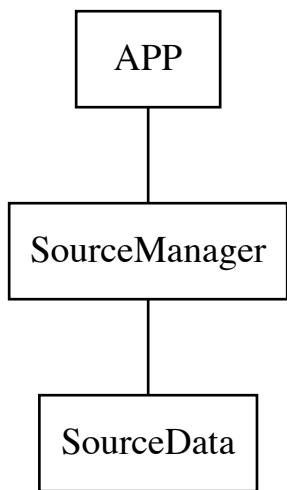
Els 3 elements claus de twain són:

1. **Application Software:** S'ha de modificar l'aplicació per treballar amb twain. És l'aplicació base per parlar i instanciar twain.
2. **Source Manager Software:** Aquest software proporciona les interaccions entre l'aplicació i la font. Proporciona el kit d'eines per desenvolupadors twain.
3. **Source Software:** És el software que controla el dispositiu d'adquisició d'imatges i està preparat per funcionar en el desenvolupament del dispositiu per complir les especificacions que marca twain. A nivell de dispositiu. S'accedeix i es gestiona un o diversos dispositius d'obtenció d'imatges.



### 7.3.1 Arquitectura

La transferència de dades és possible gràcies als 3 elements claus de twain comentats en l'apartat anterior. Aquests elements utilitzen una arquitectura definida per comunicar-se, que té l'aspecte de la figura:

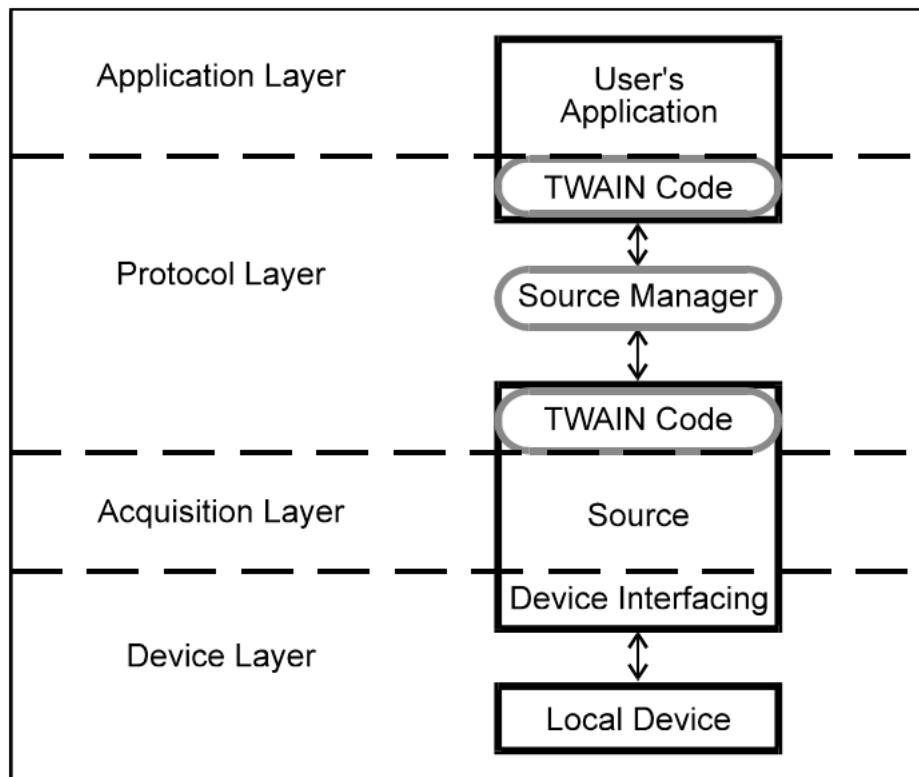


Aquesta arquitectura ocupa 4 capes:

1. Aplicació
2. Protocol

3. Adquisició
4. Dispositiu

Els elements de twain ocupen aquestes capes tal com mostra la següent figura:



Cada capa es descriu en les seccions que segueixen.

## Aplicació

En aquesta capa s'executa el software d'aplicació d'usuari. Twain defineix les directrius de la interfície d'usuari per desenvolupar aplicacions respecte a com accedir a les funcionalitats de twain, però en cap moment marca com implementar l'aplicació, sinó com interactuar amb twain.

## Protocol

El llenguatge parlat i la sintaxis utilitzada per twain. Implementa les comunicacions per cada transferència de dades. La capa de protocol inclou:

- Interfície de comunicació entre el software i twain
- Administració de Source Manager
- Administració dels Source Data

### Adquisició

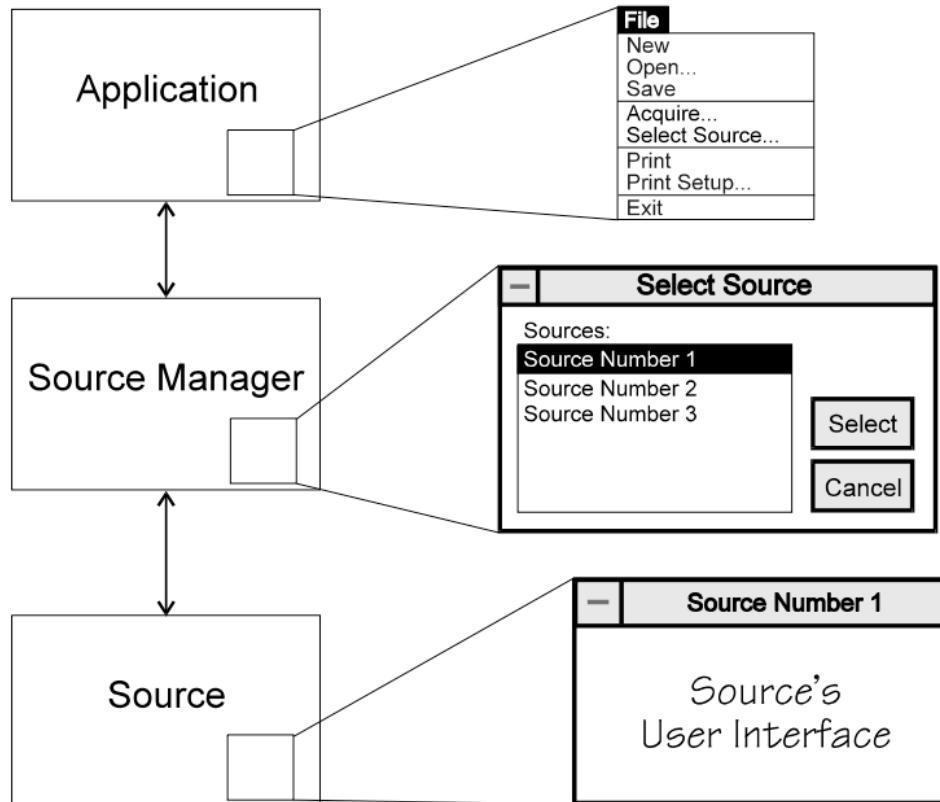
Els dispositius d'adquisició poden ser físics (p.ex: escàner) o lògics (p.ex: una imatges de base de dades). Els elements de software per controlar aquests dispositius s'anomenen fonts i són els membres principals d'aquesta capa. Aquesta capa presenta opcionalment una interfície d'usuari de control del dispositiu. També es pot presentar una propia interfície d'usuari, o simplement cap. En aquest projecte s'utilitzen varíes formes de fer-ho.

### Dispositiu

És on resideixen els drivers dels dispositius. Converteixen els dispositius específics de hardware en accions específiques pel dispositiu en particular. És una traducció HW-SW. Les aplicacions que utilitzen twain no necessiten enviar els controladors del dispositiu, ja que són part de la font (Source Data), però si que permet treballar directament amb ells.

### Interfície

Quan es construeix un software que utilitza twain, el procés d'adquisició és similar al de la figura:



### 7.3.2 Ajuda de twain en el projecte

El que proposa la figura vista anteriorment és que l'aplicació parli directament amb el SourceManager. El que es proposa és obrir un SourceManager, aquesta interfície permet llistar els source data, obtenir disponibilitat i característiques dels source data, definir el sistema de finestres a utilitzar... Amb aquest Source Manager es pot obrir i treballar directament amb un varis Source Data.

Per veure un exemple molt bàsic d'utilització, primer fixem-nos en el següent codi:

```
import twain
sm = twain.SourceManager(0)
sd = sm.open_source()
```

Aquest tall de codi importa la llibreria, obté un Source Manager, i llança un prompt a l'usuari per seleccionar l'escàner amb el qual vol treballar. Òbviament és un exemple molt bàsic “d'instanciació” per començar a treballar amb un escàner, i el mètode de creació d'un Source Manager permet moltes opcions de configuració com el nivell de finestres, el protocol... I el source data compta amb molts més detalls.

**nota:** A partir d'aquest punt sm = Source Manager, sd = Source Data.

Per una explicació bàsica, cal apuntar que sempre obrirem un Source Manager per treballar, i si és el cas d'obrir un Data Source, és convenient tancar-lo a l'hora d'utilitzar-ne un altre, o quan s'ha acabat de treballar. Seguint en l'exemple anterior:

```
sm.close()
sm.destroy()
```

Aquests dos mètodes permeten tancar el Source Data. És important comentar que twain treballa amb el sistema d'estats comentat anteriorment, i que per tant és molt important respectar l'ordre i les normes per canviar d'un estat a un altre. Altrament la llibreria llança una excepció d'error de seqüència.

### Obtenir una llista de Source Data

Ex:

```
sm.GetSourcesList()
```

Retorn: [ ] o Exception: TWCC\_NODS Retorna una llista de source datas o una excepció si no hi han Data Sources disponibles.

### Adquisició

Hi han diverses maneres d'obtenir imatges des de la llibreria de twain, principalment en el projecte s'han enfocat 3 de diferents, algunes més complexes d'altres. A tall d'exemple es comenta una de les més bàsiques, pressuposant que en l'estat de l'aplicació està en un nivell correcte, que els estats de twain romanen en el punt adient, que els Data Sources tenen les dlls necessàries per treballar, que tenim un DSM instal·lat, que hem configurat un Source Manager... Com es pot comprovar el procés d'interacció amb twain és complicat i farragós. Presuposant tot això i un munt de coses més vegem-ne un exemple bàsic

```
sm.open_source(sm.GetSourcesList() [0])
sd.RequestAcquire(1, 0)
rv = sd.XferImageNatively()

if rv:
    (handle, count) = rv
twain.DIBToBMFile(handle, path)
ret_value = handle

sm.close()
sm.destroy()
sd.close()
sd.destroy()
(sm, sd) = (None, None)
```

Un resum ràpid de què aconsegueix aquest tall de codi és: obrir una font i fer peticions twain per fer una adquisició. Existeixen varis tipus de peticions, en aquest exemple es fa una petició d'adquisició mostrant un prompt a l'usuari, i s'indica a twain que l'aplicació està llesta per digitalitzar. Cal anotar que és vital respectar sempre els estats de twain. Com aquesta hi han moltíssimes formes de fer una adquisició. També es poden utilitzar *callbacks* i subscriren-s a twain, i rebre informació dels events o canvis d'estat a través d'ells. Per certs motius com la digitalització nativa, la digitalització sense controladors o amb preferències, s'han utilitzat diversos mètodes per l'adquisició de documents. En aquesta part no es contempla, però la llibreria twain pot ser tan aprofundida com es vulgui, i s'han utilitzat moltes funcions per utilitzar preferències d'escaneig, formats...

### Subscripció d'events

Permet subscriure's als events o canvis d'estat:

```
def subscribe_twain(event):
    try:
        if event == twain.MSG_XFERREADY:
            sd.ProcessXFer()
        elif event == twain.MSG_CLOSEDSREQ:
            sd.close()
            sd.destroy()
            sd = None
    except:
        import sys, traceback
        ei = sys.exc_info()
        traceback.print_exception(ei[0], ei[1], ei[2])

sm.SetCallback(self.subscribe_twain)
```

Es pot subscriure a qualsevol nivell de missatges. És interessant, ja que twain està molt acostumat en treballar amb l'ús d'excepcions. I en el marc d'aquest treball, s'han de controlar totes les possibles accions o previsions, encara que siguin d'usuari. A tall d'exemple imaginem-nos que l'usuari cancela la digitalització, o tanca l'escàner, o... Twain envia informació a nivell d'excepcions, i per tant l'aplicació podria deixar de funcionar en qualsevol moment. Per tant aquesta funció ajuda molt a controlar el que ha de fer a cada cas, altrament la casuística augmenta considerablement, fent que sigui complicat contemplar el gran nombre de

casos. Cal fer notar que en aquest TFG no només s'utilitza twain per digitalitzar, sinó per un munt de coses més, definir preferències, obrir Data Sources, analitzar compatibilitats amb twain, explorar característiques dels Data Sources, anàlisis de versions... Òbviament existeixen altres maneres de subscriure's als events o capturar-los.

## Capabilities

Existeixen diversos tipus de capabilities per configurar característiques amb la següent nomenclatura:

- CAP: Capabilities Generals
- ICAP: Capabilities d'Imatge
- ACAP: Capabilities d'Audio

Comprovar capabilities:

la funció *GetCapability* permet analitzar una propietat específica segons un Source Data donat:

```
self.sd.GetCapability(twain.ICAP_PIXELTYPE)
```

Aquesta funció s'utilitza per retornar l'informació de la capability del Data Source. Si no s'admet la capability retorna una excepció. El tipus de retorn és:

```
Tupla del tipus (TWTY_ *) i un valor.
```

El format dels valors depèn del tipus de contingut. Les caps poden estar en els següents:

```
singleton, range, enumerator or array
```

ex:

```
(1, (0, 0, [1, 2, 4]))
```

L'últim array indica els valors permesos de la capability. Seguint l'exemple els valors assignables a la capability són: TWPT\_BW, TWPT\_GRAY, TWPT\_PDF, ja que 1, 2, 4 són el valor d'aquestes respectivament

CAPS:

```
twain.TWPT_BW  
twain.TWPT_GRAY  
twain.TWPT_RGB
```

## 7.4 PyQt

Per presentar una vista de l'aplicació s'ha utilitzat l'entorn de treball PyQt, que implementa els estàndards Qt amb Python. Qt està fet amb C++, i PyQt permet la utilització de Qt amb el llenguatge Python.

És interessant escollir un bon sistema de vista o GUI. Les raons perquè s'ha escollit Qt són les següents:

- Multisistema

- Utilitza elements natius de cada sistema \*
- Fitxer xml per estils
- Software de designer
- Molta documentació
- Suport
- Utilitza els elements de finestres, estils colors... de cada sistema

A part del gran suport que té Qt, s'utilitza per desenvolupar software fiable i que s'ha anat fent popular. S'utilitza per dissenyar smart tv's, ordinadors a bord d'alguns cotxes, smartwatches i per descomptat aplicacions, moltes aplicacions. Si els més grans l'escullen, serà per alguna cosa. I els avantatges que s'han analitzat per utilitzar Qt com a sistema GUI en són moltes, algunes esmentades a la llista anterior.

Òbviament PyQt té molta documentació com s'ha esmentat, però a més a més és una gran sort que, sigui un sistema tant utilitzat, ja que de forma molt ràpid algú et pot donar suport. D'exemple es pot veure la Issue que es va obrir per un dubte d'aquest projecte, i que es va resoldre de forma molt ràpida:

<https://stackoverflow.com/questions/43913716/pyqt5-focus-qmainwindow-as-first-window>

Quan es desenvolupa en un sistema GUI, el mètode de treball sol ser similar per tots, però el que el fa característic, és la facilitat o no d'accés, i la visibilitat.

**Fitxers .ui:** La responsabilitat més important del programador és que la vista sigui funcional, queda més en segon pla o com a segona tasca el disseny. Preneix més importància la funcionalitat dels elements i menys el disseny de tals. Això és degut a que un element p.ex: del tipus botó, pot ser molt agradable per la vista, però si no fa res, o no funciona, o no fa el que hauria de fer, no serveix per res. Per tant interessa centrar-se més en la funcionalitat, i una de les eines que es poden utilitzar en PyQt és *QtDesigner*, que abstreueix al programador de dissenyar una interfície de vista purament per codi, sinó que permet dissenyar a mode gràfic una vista agradable, i posteriorment carregar-la al programa i encarregar-se de la funcionalitat dels elements ara si via codi.

PyQt treballa amb una herència molt definida, posa a la disposició elements abstractes, i elements cada cop a nivell més concret que hereden d'aquests primers.

Existeixen diverses versions de PyQt, i un dels canvis més notoris entre versions és la separació dels *packages*. S'ha utilitzat la versió 5. Els elements de Qt es divideixen en 4 grans *packages* segons l'accés i la funcionalitat:

- Core: nucli de Qt, accés a característiques i definició de posicionats d'elements...
- Ui: accés a fitxers .ui, i càrrega dels mateixos
- Gui: accés a interfície bàsica, i pintat de marcs
- Widgets: accés a elements visuals de la vista (botons, menús, layouts...)

PyQt treballa amb una estructura de comunicació típica de les GUI's. Els usuaris produueixen una acció, l'element capta tal acció i reacciona. Aquesta metodologia es sol treballar a través de callbacks i senyals. A PyQt es parla de senyals i slots.

- Senyals: són les senyals que emeten els elements al produir-se una acció
- Slots: són les funcions que es connecten a les senyals

connexió de botons: com s'ha mencionat abans, podem connectar elements de la vista amb accions. L'acció del clic d'un botó es connecta a una funció de la següent manera:

```
self.name_button.clicked.connect(self.name_function)
```

Pot ser interessant conèixer propietats del botó o reaprofitar una funció en varis botons. Podem fer ús del self.sender() En aquest exemple pràctic es mostra com determinar quin botó ha enviat la senyal

```
self.button_1.clicked.connect(self.name_function)
self.button_2.clicked.connect(self.name_function)

def name_function(self):
    print(self.sender())
```

També es permet passar paràmetres directament amb l'ús de les lambdes de python

```
self.button_1.clicked.connect(lambda: self.name_function(1))
self.button_2.clicked.connect(lambda: self.name_function(2))

def name_function(self, numer):
    print("Button ", number, " clicked")
```

Altres elements com Combobox, TableView, ListView... es poden connectar a senyals. Combobox: ens serà interessant saber quan i si hem canviat un element del nostre combobox.

Sovint és interessant realitzar un procés de *customització* de una GUI mitjançant codi. PyQt posa a disposició una sèrie de funcions per tal de estilitzar una vista. Cuidant molt l'herència i delegant responsabilitats s'ofereixen mètodes per afegir, decorar, eliminar... components. Cada canvi té assignat un responsable, és a dir que el canvi de un component s'haurà de fer sobre el mateix. Un possible exemple és el que segueix: Quan s'intenta canviar el *header* d'una taula, s'atacarà directament sobre aquest *header* i no sobre la taula. A continuació es detalla en aquest fragment de codi:

```
self.table_name()
self.table_name.horizontalHeader().setSectionResizeMode(QHeaderView.Fixed)
```

En l'exemple vist, es canvia el mode en el qual els elements d'un *header* canvien de tamany. En aquest cas es posa a fixed, fent que l'usuari no pugui fer *resize* directament a la vista. Però l'interessant és veure com es deleguen les responsabilitats als elements de Qt. Com que es canvia una propietat del *header* s'ataca directament al *\_horizontalHeader\_*, y no a la taula. I la propietat corresponent és pertanyent a la vista d'un header, per tant es passa a la funció un element del tipus QHeaderView.

PyQt té un munt de coses interessants, per exemple a comentar. En aquest software s'utilitza una icona de sistema per donar un feedback a l'usuari de que el programa està corrent i que tot està correcte. PyQt posa a la disposició un SystrayIcon.

## 7.5 ØMQ

Després d'un estudi i petit disseny de la comunicació, es va decidir utilitzar sockets per una comunicació del tipus IPC. El llenguatge Python incorpora una llibreria de sockets, però ja que aquest projecte té previsió de desplegar-se en diverses empreses, dins de les quals es mouran forces dades, les mètriques de temps podrien

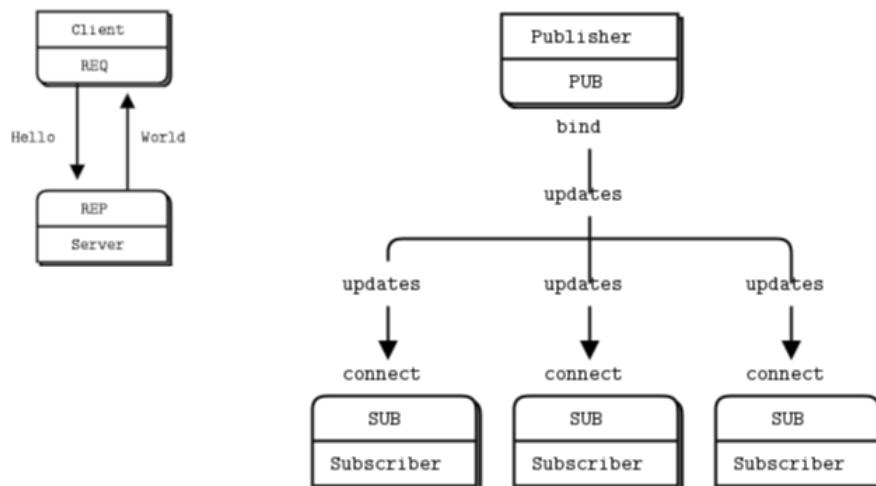
augmentar considerablement. I una bona solució és utilitzar una plataforma per la comunicació via sockets molt ràpida.

ØMQ és una plataforma que segons <http://zeromq.org/> permet:

- Connectar el codi en qualsevol llenguatge i en qualsevol sistema.
- Utilitzar les arquitectures: IPC, TCP, TIPC, i multicast.
- Ús de patrons pub-sub, push-pull i router-dealer.
- Motors E/S asíncrons d'alta velocitat
- Construir qualsevol arquitectura: centralitzada, distribuïda, petita o gran.
- Software lliure

Es proposa la meva socket-client i també la creació d'un socket PUBLISHER, i altres SUBSCRIBERS, que es subscriuren a aquest PUBLISHER, però que a més es poden subscriure a certs nivells de missatges, i per tant els sockets SUBSCRIBER només recolliran els missatges necessaris.

És interessant en aquest treball utilitzar dues metodologies de sockets, que en propers capítols es detallaran. Un d'estrucció client - servidor, i l'altre publisher-subscribers



ØMQ conegut també per zmq, facilita molt l'enviament de dades en format tipus JSON. Actualment s'està tornant un format molt estàndard, i zmq dóna les facilitats d'enviar un paquet en del tipus, string, llista, diccionari... i parsejar-ho com a JSON. A l'altra banda del socket es podrà rebre en el format que es necessiti. És un avantatge poder fer una bona abstracció de dades gràcies a facilitats com la d'enviament de dades en format JSON, ja que no tots els sockets en altres versions no ho permeten o és de difícil utilització.

El llenguatge Python pot incorporar la llibreria `pyzmq` que proporciona l'ús i l'accés a zmq.

Anteriorment s'ha comentat que són dos les arquitectures de sockets que s'utilitzaran en aquest projecte. A mode d'exemple a continuació es mostra la creació d'un socket **REP** d'arquitectura REQ - REP, que s'ha utilitzat en el projecte:

**Nota:** El següent bloc de codi utilitza la importació de @IP, #PORT i Protocol des d'un fitxer de configuració, per tal de que sigui més fàcil la configuració i el desplegament de l'aplicació en diferents entorns.

```

import configparser
import pyzmq

@property
def route(self):
    parser = configparser.ConfigParser()
    parser.read('config.cfg')
    port = parser.getint('PORT', 'port')
    ip = parser.items('HOST')[0][1]
    protocol = parser.items('PROTOCOL')[0][1]

    return '{0}://{}:{1}'.format(protocol, ip, port)

def sock_connect(self):
    context = zmq.Context()
    self.publisher = context.socket(zmq.PUB)
    self.publisher.bind(self.route)

```

## 7.6 SQLite

Cal gestionar una base de dades per guardar les preferències i els perfils que gestionin els usuaris. SQLite (<https://www.sqlite.org/>) és una base de dades relacional que proporciona l'emmagatzemament de la base de dades en un fitxer que conté la definició de l'estructura de les taules i les dades. Es proposa un sistema eficient i *light*, ja que es necessita una base de dades lleugera, on el volum de dades a emmagatzemar és relativament petit. A més, no és un sistema de gestió de base de dades que funcioni amb una metodologia client-servidor, sinó que està integrat totalment dins el programa. Per tant a l'hora d'analitzar els requisits necessaris per la creació i gestió de la base de dades, SQLite s'ajusta molt bé a les necessitats proposades.

Existeix una llibreria de Python anomenada sqlite3, per gestionar una base de dades amb SQLite en aquest llenguatge. A continuació es detalla el funcionament bàsic:

Es proposa crear un connector a la base de dades i un cursor per realitzar operacions sobre aquesta. La base de dades està persistida en un fitxer **.db**. Si la base de dades no existeix, SQLite la crea automàticament.

```

import sqlite3
conn = sqlite3.connect('foo.db')
cursor = conn.cursor()

```

### Create table:

```
cursor.execute('''Create table if not exists Foo (id_foo, name)''')
```

Amb `create table if not exists`, ens assegurem que que no es llanci una excepció de taula ja existent. Després de cada operació, o de varies, es graven els canvis utilitzant el connector:

```
conn.commit()
```

**Inserts:** Es permeten inserts unitaris o multiregistre.

```
cursor.execute(''Insert into Foo values (1, 'name')'')
conn.commit()
_values = [(1, 'name_1'), (2, 'name_2'), (3, 'name_3')]
cursor.executemany('Insert into Foo values (?, ?)', _values)
conn.commit()
```

**Query:** Les consultes són del tipus registre unitari o multiregistre.

```
cursor.execute(''select * from Foo'')
result = cursor.fetchone()

cursor.execute(''select * from Foo'')
result = cursor.fetchmany()

'''equivalent al cursor.fetchmany'''
for row in cursor.execute(''Select * from Foo''):
    print(row)
```

Vegem-ne una petita utilització en el projecte:

```
def insert_one(self, field, value):
    try:
        query = '{0}{1}{2}{3}{4}'.format(''Insert into preferences (''', field, ''') values("'''', value, '''")''')
        self.cursor.execute(query)
        self.conn.commit()

        self.logger.info(self.cursor.execute('select * from preferences').
                         fetchone())
    except sqlite3.OperationalError:
        self.logger.info('##### NO INSERT FIELD')

def update_one(self, field, value):
    try:
        query = '{0}{1}{2}{3}{4}'.format(''update preferences set ''', field,
                                         '''=''', value, '''''')
        self.cursor.execute(query)
        self.conn.commit()

        self.logger.info(self.cursor.execute('select * from preferences').
                         fetchall())
    except sqlite3.OperationalError:
        self.logger.info('##### NO UPDATE FIELD')
```

## 7.7 cx\_Freeze

Aquesta llibreria permet generar executables en llenguatge Python. Aquest projecte es desplegarà en un Sistema Operatiu Windows, i `CX FREEZE` ajuda a crear fitxers `.exe` o instal·ladors `msi`.

Per utilitzar aquesta eina es pot instal·lar via pip igual com s'ha mostrat amb altres llibreries d'aquest capítol.

```
pip install cx_Freeze
```

Aquesta eina legeix un fitxer de setup: `setup.py`, i genera l'executable. Per tant cal crear un fitxer amb l'estructura que proposa cx\_Freeze.

La forma més senzilla de generar un primer setup és utilitzar la comanda:

```
cxfreeze-quickstart
```

Ens demanarà el nom del projecte, la versió... En realitat l'únic que fa és crear un arxiu `setup.py` amb la versió, nom... Però és interessant saber que en aquest fitxer s'han d'incloure totes les dependències, paquets, paths... Per fer-ho cal seguir l'estructura que proposa CX FREEZE. Un cop programat el setup, només cal executar-lo amb les opcions que es necessiten:

```
python setup.py [options]
```

Opcions interessants:

#### Construir un executable:

```
python setup.py build
```

#### Construir un instal·lador:

```
python setup.py bdist_msi
```

En el projecte interessa construir un instal·lador.

Setup actual

```
from cx_Freeze import setup, Executable
import sys
import os
base = 'Win32GUI' if sys.platform == 'win32' else None

shortcut_table = [
    ("DesktopShortcut",           # Shortcut
     "DesktopFolder",            # Directory_
     "ScannerApp Connector",    # Name
     "TARGETDIR",                # Component_
     "[TARGETDIR]connector.exe", # Target
     None,                      # Arguments
     None,                      # Description
     None,                      # Hotkey
     None,                      # Icon
     None,                      # IconIndex
     None,                      # ShowCmd
     'TARGETDIR'                 # WkDir
   )
]

# Creació de la taula d'accisos
msi_data = {'Shortcut': shortcut_table}
```

```
# Change les options MSI per defecte i especifica l'ús de la taula definida
bdist_msi_options = {'data': msi_data}

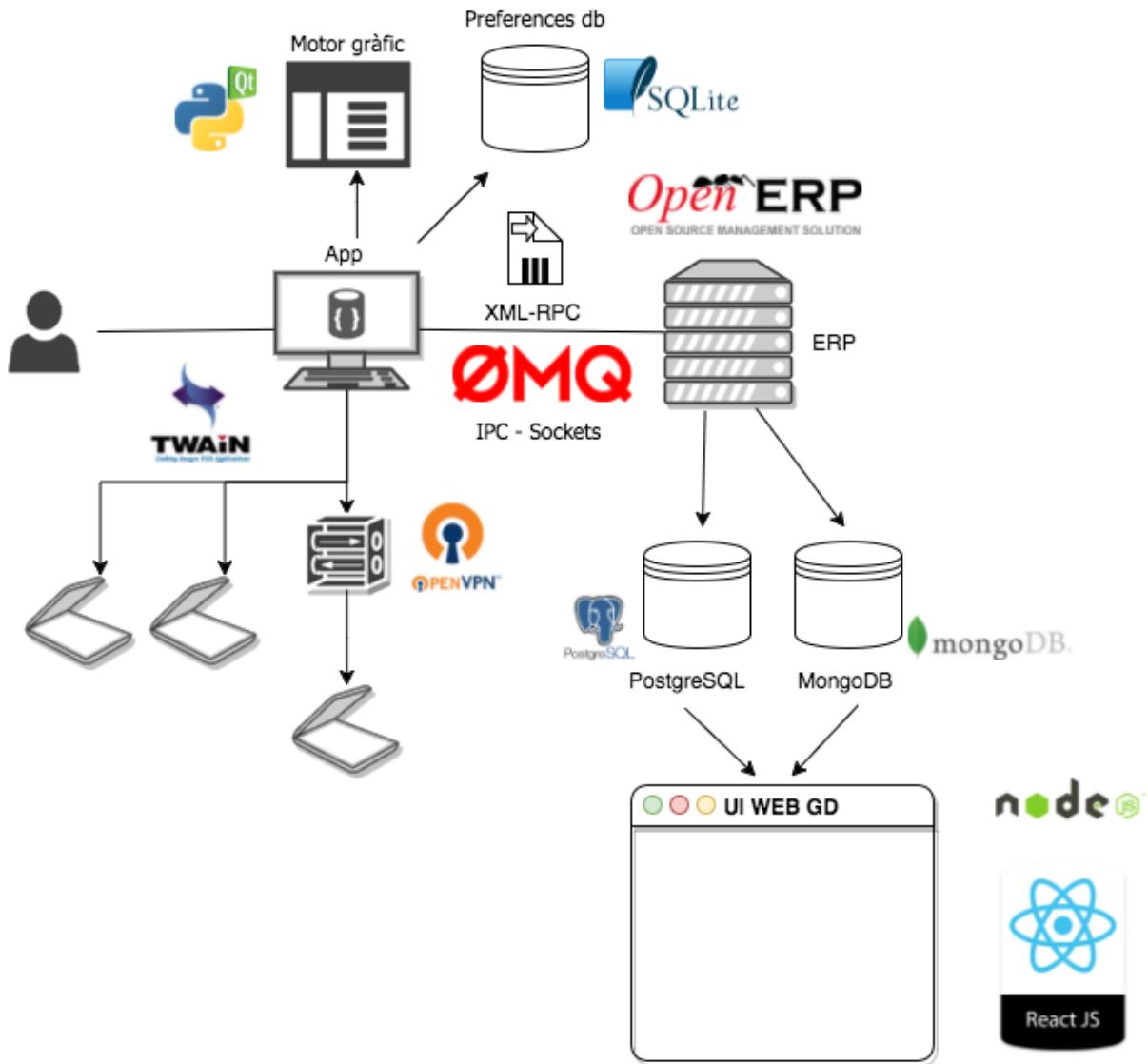
options = {
    'build_exe': {
        'include_files': [
            'scannerapp/GUI/templates',
            'scannerapp/GUI/images',
            'scannerapp/agent/config.cfg',
            'scannerapp/GUI/images/printer.png',
            os.path.join(sys.base_prefix, 'DLLs', 'sqlite3.dll'),
            os.path.join(sys.base_prefix, 'DLLs', 'tk86t.dll'),
            os.path.join(sys.base_prefix, 'DLLs', 'tcl86t.dll'),
        ],
        'packages': [
            'PyQt5.uic',
            'PyQt5.QtGui',
            'PyQt5.QtWidgets',
            'PyQt5.QtCore',
            'PIL',
            'zmq',
            'os',
            'tkinter',
            'raven',
        ],
        'excludes': [
            'django',
            'Image',
            'PyQt5.Qt',
        ],
    },
    'bdist_msi': bdist_msi_options,
}

executables = [
    Executable(script='scannerapp/GUI/GUI.py', base=base),
    Executable(script='scannerapp/agent/connector.py', base=base)
]

setup(
    name='ScannerApp',
    version='0.2.5a0',
    description='Scanner application integration',
    options=options,
    executables=executables,
    author='GISCE-TI, S.L.',
    license='MIT',
    skip_build=True
)
```

## 7.8 Diagrama bàsic de funcionament

Exposades les llibreries, els conceptes que engloben el marc d'aquest TFG, i les eines utilitzades es mostren en la següent figura:





## Anàlisi i disseny del sistema

---

### 8.1 Estructura modular

Una de les responsabilitats més importants del programador és generar un codi net, ordenat i funcional. Es parla de responsabilitats degut a la necessitat de reimplementació i reutilització. Si es desenvolupa sense ocupació d'aquests detalls, el software es torna tancat. D'altra manera podem reimplementar-lo, o qualsevol en podrà reutilitzar mòduls del software. El projecte es divideix en diferents parts, cadascuna com si fos una peça de puzzle modelable. Parlem d'agent, comunicació, vista i API base, que es tractaran en capítols posteriors.

### 8.2 Anàlisi de Responsabilitats

La primera fase d'anàlisi de l'aplicació és vital pel desenvolupament. Tot projecte, per petit que sigui requereix una fase d'anàlisi i disseny posterior abans de passar a la fase d'implementació, i encara més en un TFG.

Per tal que les funcionalitats estiguin en mòduls separats i que el software es torni modelable, s'analitzarà des d'un punt de vista de canvi. És a dir, el que es va fer va ser el següent: Pensar i analitzar quines peces podrien canviar-se en un futur o haver de funcionar per si soles. El motiu de proposar un canvi en un futur no és perquè es vulgui canviar. No tindria sentit desenvolupar de bon principi una aplicació d'una manera que saps que en breus en canviàrs la implementació. El sentit és més per si es vol distribuir el software independentment, per si algun sistema passa a *depercated*, per si algú en vol utilitzar tan sols una part... Com s'ha esmentat en capítols anteriors, existeixen gestors documentals, però es fa difícil trobar-ne en obert, i que siguin tan adaptables com aquest. Per fer-se una idea amb les "peces" que es llisten a continuació, qualsevol pot agafar la part de l'api i construir-se la seva vista amb el seu programa, o simplement utilitzar el software implementant la seva comunicació, definir-se una API pròpia, o integrar-ho en un altre ERP o software que desitgi entre d'altres. Per tant analitzant quines funcionalitats podrien haver de treballar independentment o canviar-se, ens surt una llista com la següent:

- Connectar amb l'ERP
- Possibilitat de canviar la vista
- Possibilitat de canviar la comunicació
- Possibilitat de canviar l'API
- Possibilitat de canviar el protocol d'accés als escàners

A partir d'aquest punt es defineixen els mòduls que poden anar bé per aconseguir el detallat.

- Per connectar amb l'ERP: es necessita un connector, per parlar atendre a l'ERP, que en qualsevol moment li pot indicar: obre't, vull escanejar. Es proposa fer un:

### **Connector**

- Per la vista: es vol independència total, la vista no s'ha d'assabentar com es fan les coses més enllà. L'únic que s'ha d'encarregar és de pintar els events i demanar el que els usuaris indiquin a través de clics o inputs... Per tant es proposa crear una vista i separadament un client, per tal que aquest últim faci les peticions que indiqui la vista, de tal manera que la vista simplement renderitzarà allò que se li demani, i demanarà al client el que vol, sense importar-li a qui ho demana o com ho fa. A més és obvi que si s'implementa d'aquesta manera, si es vol es té la possibilitat d'usar el programa a mode consola, fent peticions al client, que és el que fa la vista. Això pot anar molt bé per fer canvis ràpids a la vista, implementar-ne una de nova, o simplement utilitzar el programa sense interfície d'usuari. Es proposa fer:

### **Client**

### **GUI**

- Per la comunicació: el Client podria fer directament peticions al protocol d'escaneig, però i si es vol canviar, o simplement es vol fer servir més d'un protocol? Per resoldre això es proposa una API, que serà fàcilment adaptable i de canvis ràpids, si el client té independència del protocol. És a dir, el client farà peticions a l'API, amb independència del protocol que s'utilitzi a l'altre extrem. D'aquesta manera l'API podrà implementar els protocols que vulgui. Es proposa un dispatcher per posar en funcionament una API.

### **Dispatcher**

- Possibilitat de canviar el protocol d'accés als escàners: Si s'implementa un mòdul intermediari de seleccionar el protocol adequat, s'aconsegueix afegir o canviar tantes vegades el protocol com es vulgui. Es proposa un mòdul encarregat de rebre peticions directament des del dispatcher i seleccionar el protocol a utilitzar. En resum, aquest mòdul podria rebre la petició d'escanejar, i triar segons els paràmetres que es vulgui el protocol d'accés a escàners, per exemple twain o sane. Per tant es proposa un mòdul agent i els altres de protocol. En aquest cas:

### **ScannAgent**

### **TwainBase**

### **SaneBase**

Un altre dels temes importants a tractar és l'adaptació del programa amb l'ERP. Per fer-ho es proposa crear una extensió a l'ERP a mode plugin per comunicar els dos programes i que treballin conjuntament. Cal recordar que per objectius el programa es comunica directament amb l'ERP, però la implementació ha fet

que sigui una opció decidible, ja que pot treballar sol, amb un ERP, o amb qualsevol altre programa de qualsevol altre sistema. Per tant s'introdueix un nou mòdul al projecte:

### **ERPlugin**

En aquest punt els mòduls queden definits, però falta associar-los i dissenyar com serà la interacció entre ells.

**ERP - CONNECTOR - Dispatcher - SCAGENT - TWBASEPROT GUI - CLIENT - CONNECTOR**

Ara tenim el qué (mòduls), el perquè (interaccions) però falta el com. En aquest punt es detalla com es comunicaran els mòduls:

Durant el disseny de l'aplicació es va decidir que seria molt interessant que certs mòduls no haguessin de córrer per força en la mateixa màquina física. D'aquesta manera es podria tenir perfectament un mòdul en una màquina, i un altre al servidor... Anàlogament una manera molt interessant de dur a terme aquesta tasca és la utilització de sockets. Per tal d'entendre amb més detall els sockets que es necessiten vegem el següent:

- Interessa separar: Interfície gràfica del procés d'escaneig
- Interessa separar: Plugin de l'erp del procés d'escaneig

El primer punt tal com es detalla en capítols anteriors és interessant per no tenir dependència de la vista, i poder interactuar amb el procés de la manera que es vulgui o es necessiti. És a dir que es podrà canviar la vista o usar el programa sense vista en qualsevol moment.

El segon punt intenta comunicar dos processos independents com és l'ERP i el programa d'escaneig i gestió documental.

D'aquests dos punts en surten dos sockets de la següent manera:

- ERP - Connector
- Connector - GUI

El connector rep aquest nom precisament per la funció que fa, connectar. És la peça clau encarregada de connectar el software del primer extrem (en aquest cas ERP) amb la interfície de l'altre (en aquest cas vista amb PyQt), i connectar les funcions del software d'escaneig i gestió documental amb la vista.

Cal recordar que, com es comenta en el capítol de requisits, l'erp guarda els documents al recurs de l'ERP en que s'està en el moment d'ençegar l'ScannerApp.

## **8.3 Disseny de l'API**

L'API ha estat dissenyada per atendre les peticions del client de la vista. Amb la interacció de l'usuari, bé sigui amb una interfície gràfica o sense, demanarà recursos, i l'API les obtindrà i les servirà.

L'API, principalment utilitza l'empaquetament de dades en format *JSON*. Està construïda amb una llibreria anomenada “werkzeug”, que a la fase d'implementació es detallarà millor.

Aquest és el disseny principal de les peticions a l'API i els respectius formats de retorn:

Llistar escàners:

```
GET /scanners
```

Return: [scanners: n] o ['NO SCANNERS']

Llistar escàner:

```
GET /scanner/<scanner_id>
```

Return: [scanner: 1] o ['NO SCANNER']

Obtenir format de sortida:

```
GET /format
```

Return: 'format' o 'ERROR'

Obtenir path per defecte on es guarden els documents a disc:

```
GET /path
```

Return: 'path' o 'ERROR'

Actualitzar o crear preferences:

```
POST /scanner/{
  scanner_default: string,
  color: string,
  dpi: integer,
  disposicion: string,
  path: string,
  delete_files: boolean
}/preferences
```

Return: 'OK' o 'ERROR'

Obtenir preferències:

```
GET /preferences
```

Return:

```
{
  scanner_default: string,
  color: string,
  dpi: integer,
  disposicion: string,
  path: string,
  delete_files: boolean
}
```

Eliminar preferències:

```
DELETE /preferences
```

Return: 'OK' o 'ERROR'

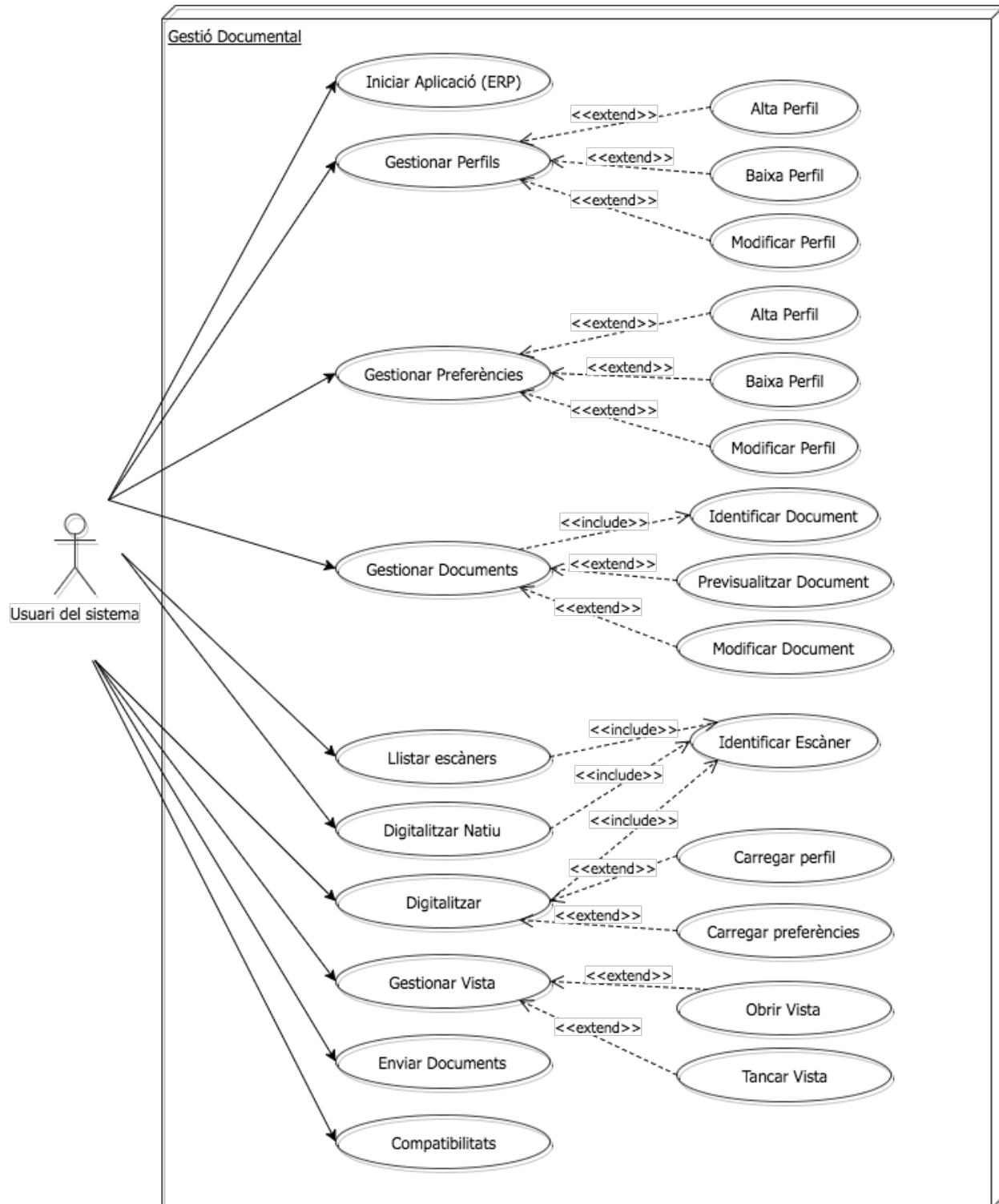
Actualitzar la id de la vista:

```
POST /windowID/<int>
```

Return: 'OK' o 'ERROR'

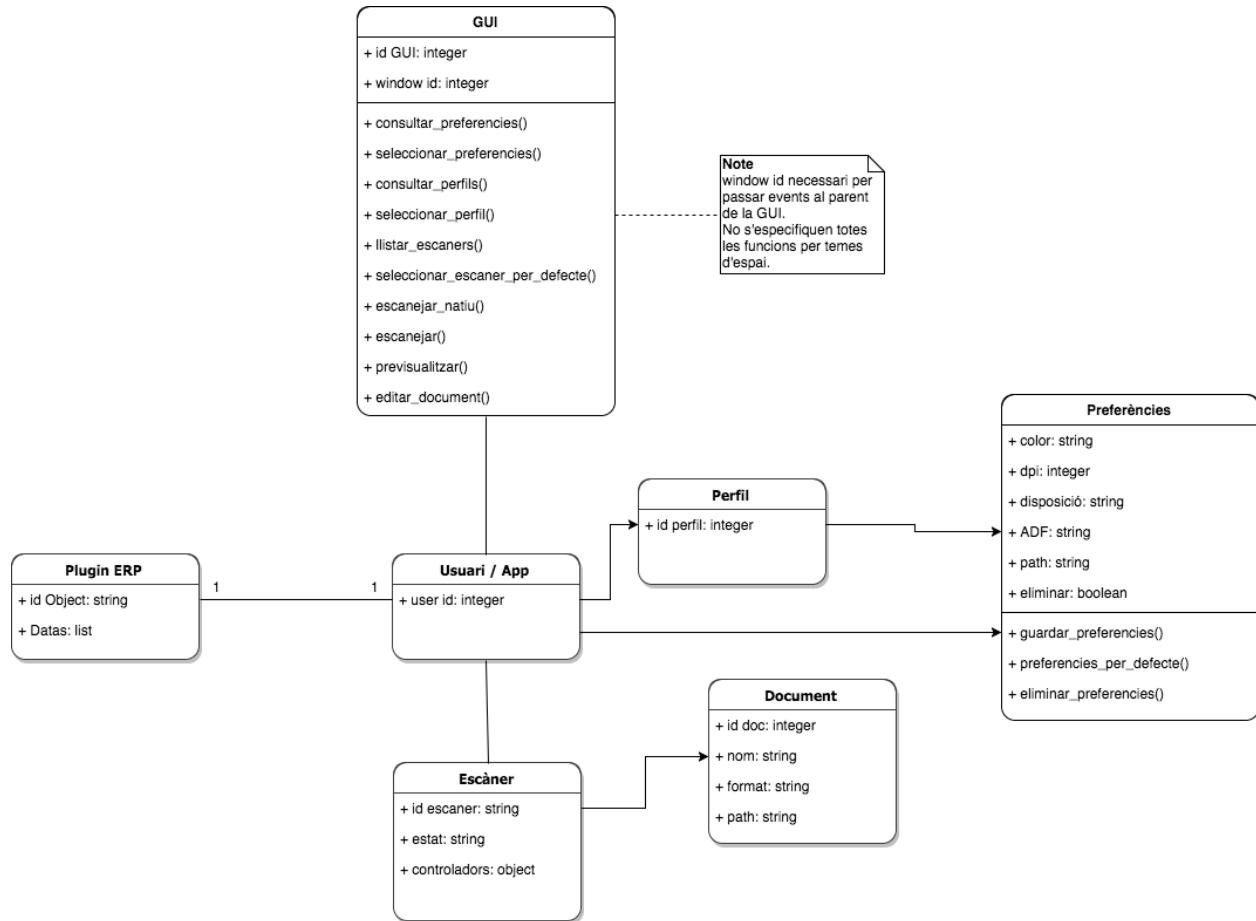
## 8.4 Diagrama de casos d'ús

Els principals casos d'ús com són: digitalitzar, gestionar preferències, gestionar documents, enviar documents... es detallen en el següent diagrama de casos d'ús de context de dos nivells d'especificació:



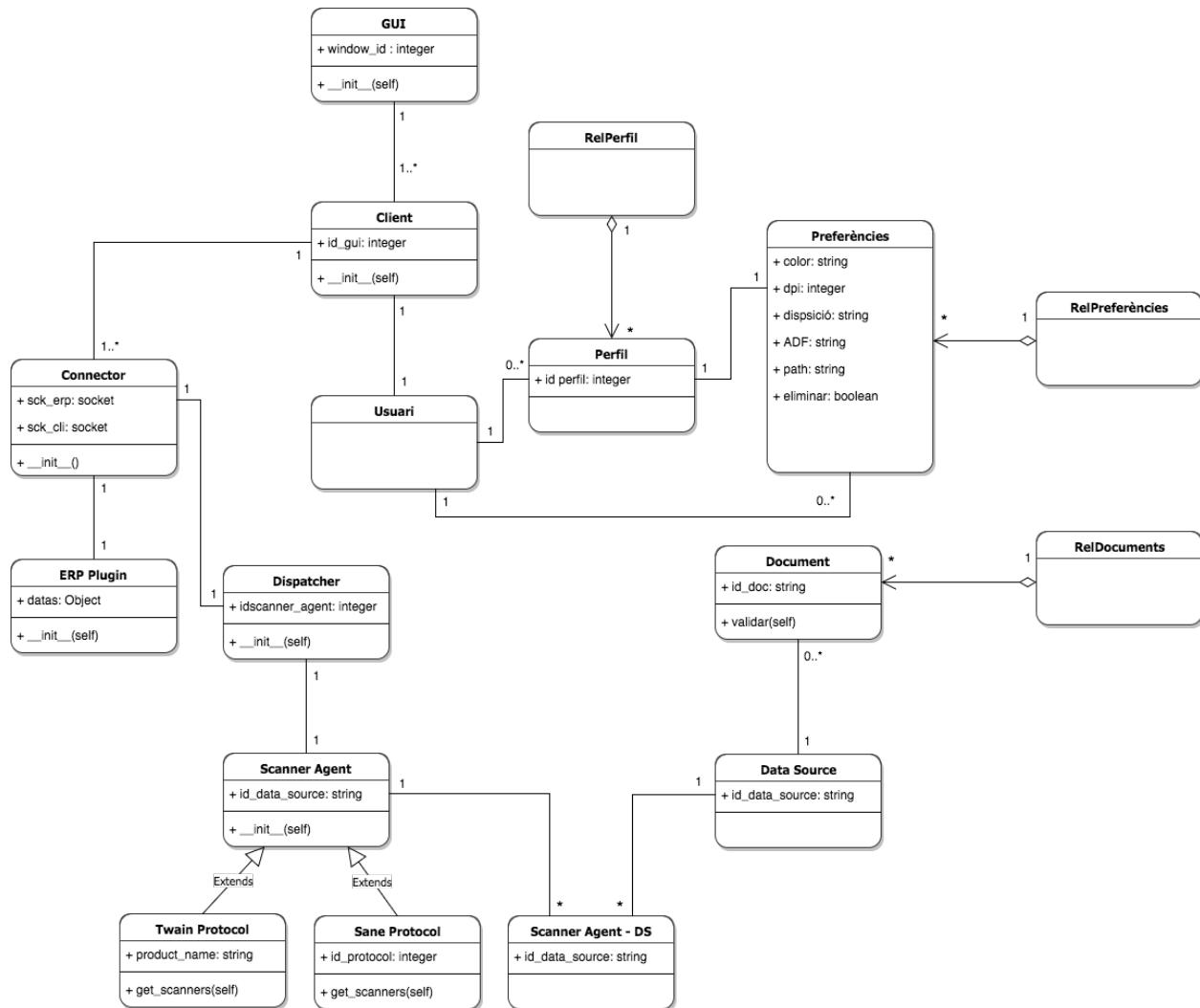
## 8.5 Diagrama de classes bàsic

En un principi es proposa un diagrama de classes bàsic, modelant-ne les classes candidates a gestionar el sistema com el següent:



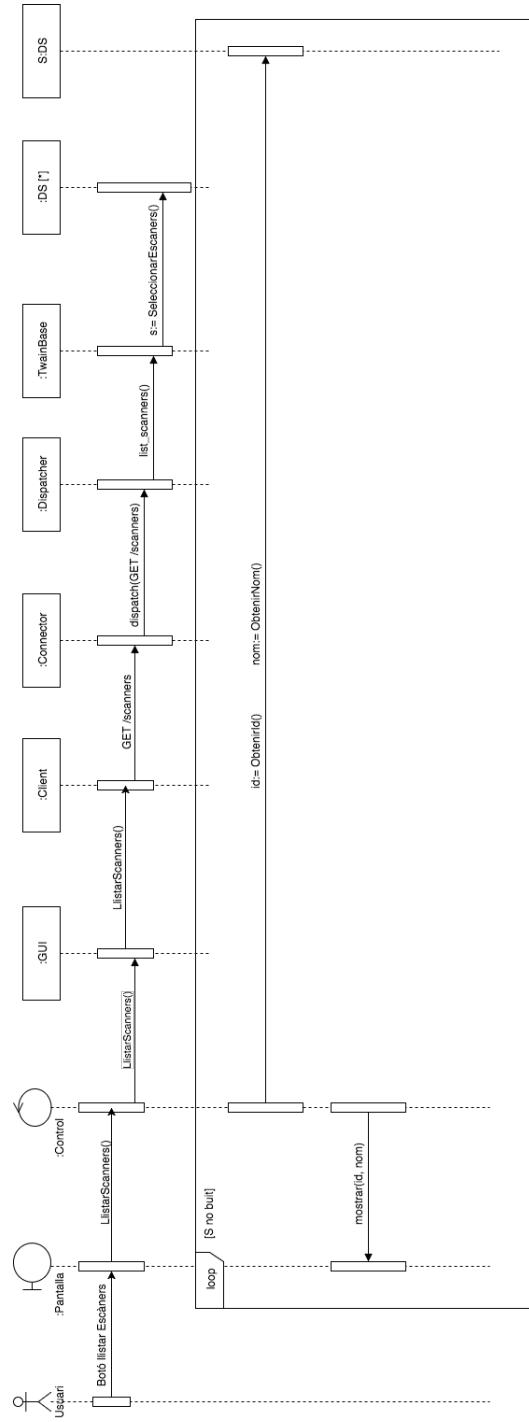
De totes maneres, tal com es comenta en apartats anteriors, es busca la implementació a mode modular i adaptativa. És per això que es fa un replantejament de l'estructura de l'aplicació segons aquestes necessitats.

## 8.6 Diagrama de classes principal



## 8.7 Diagrama de seqüència Llistar Escàners

A continuació es mostra el diagrama de seqüència **Llistar escàners**

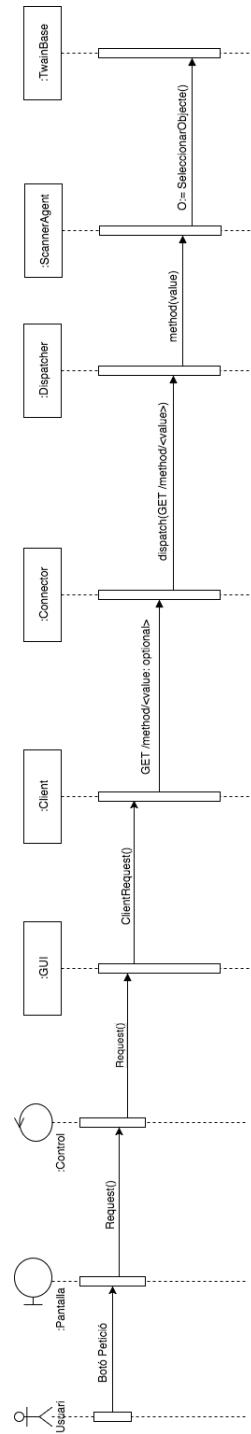


En aquest diagrama de classes es pot observar com s'utilitza una petició específica per obtenir els escàners disponibles. Aquestes peticions s'utilitzaran freqüentment en el sistema. Per no reescriure cada cop aquestes peticions, s'ha creat un diagrama de seqüència per peticions genèriques. UML posa a disposició els diagrames SM, per tal d'incloure fragments de diagrames dins d'altres per tal d'aprofitar espai i poder donar

una visibilitat i comprensió més bones. Així podrem utilitzar aquest diagrama en altres utilitzant el requadre *SD* per inclore'l. Per donar una referència el diagrama ha de tenir un nom, en aquest cas és el **Request**:

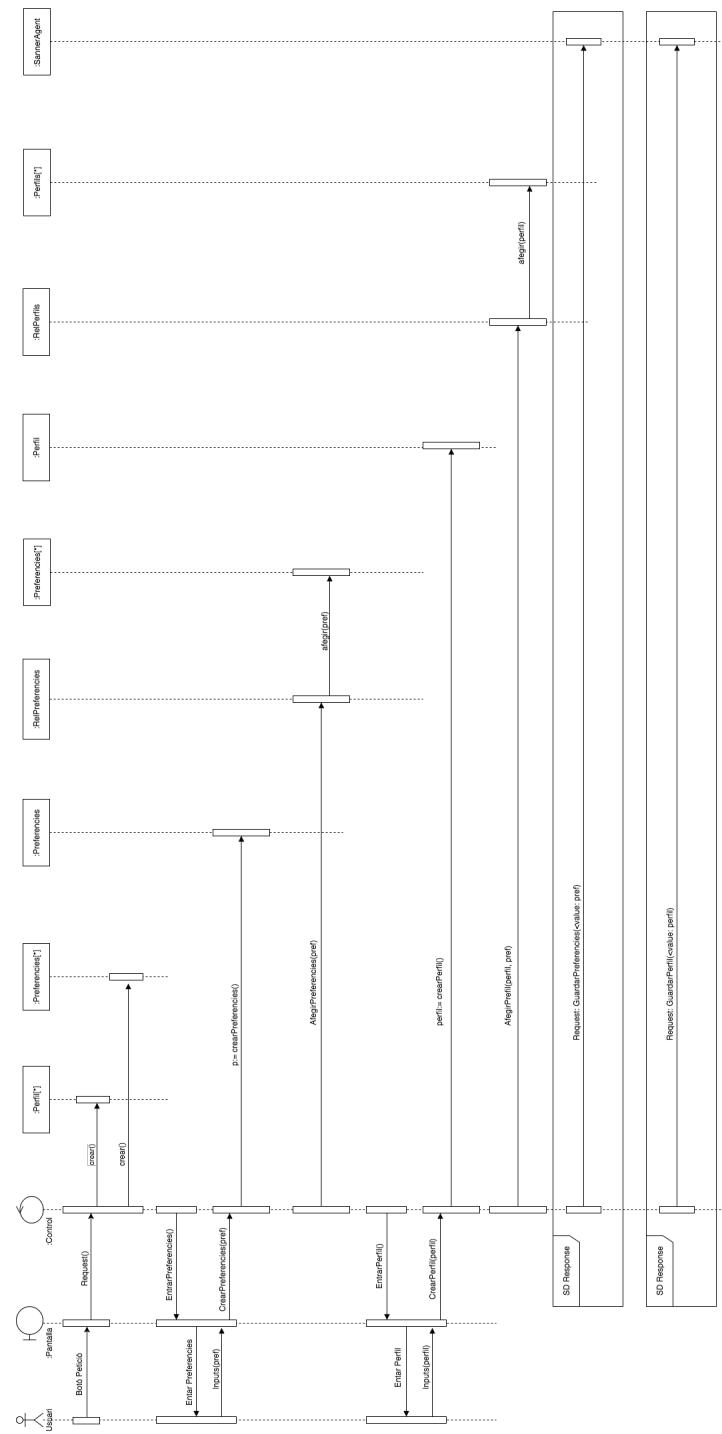
## 8.8 Diagrama de seqüència Request

A continuació es mostra el diagrama de seqüència **Request**



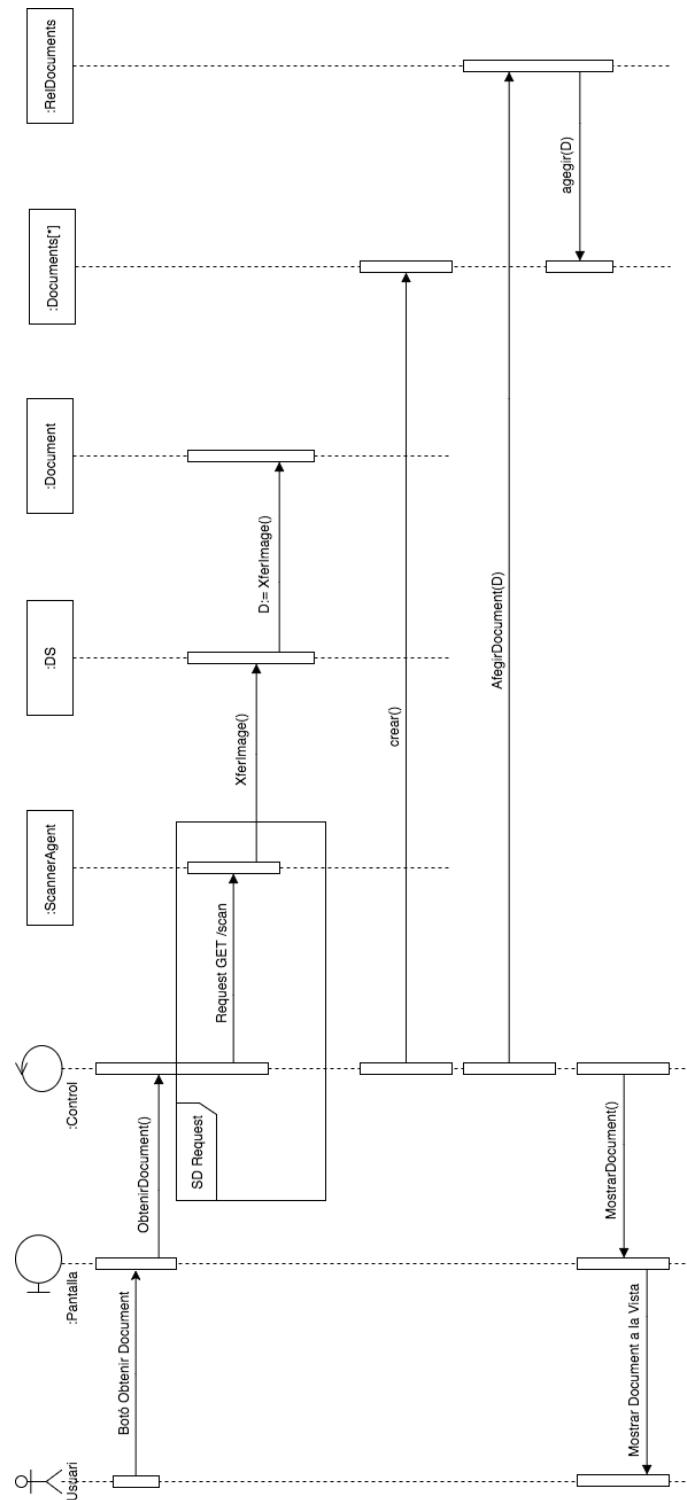
## 8.9 Diagrama de seqüència Crear perfil

A continuació es mostra el diagrama de seqüència **Crear perfil**



## 8.10 Diagrama de seqüència Obtenir document

A continuació es mostra el diagrama de seqüència **Obtenir document**



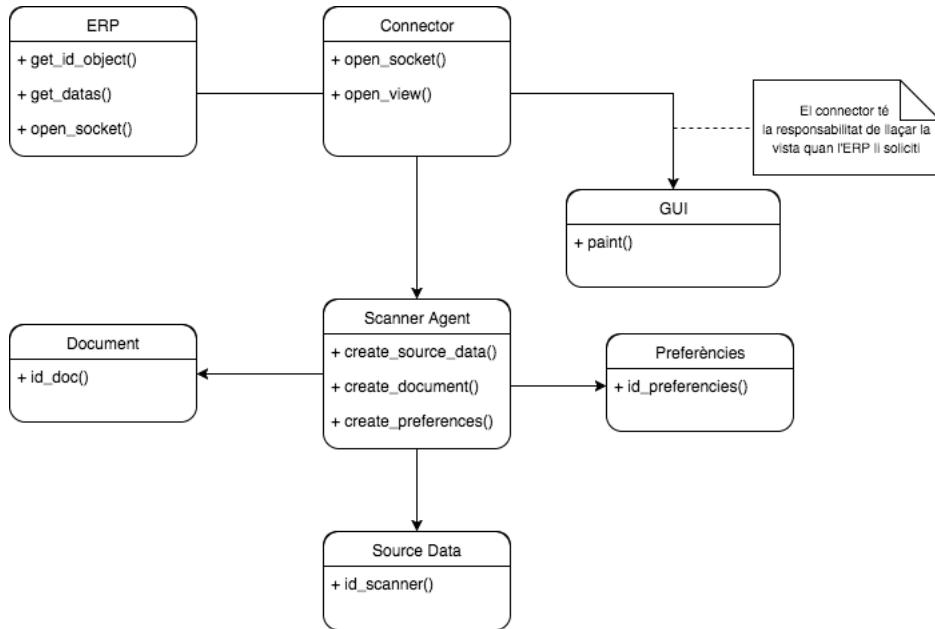
## **8.11 Assignació de responsabilitats**

Per fer un anàlisi exhaustiu de com desplegar aquests mòduls, s'ha utilitzat el mètode d'assignació de responsabilitats, que consisteix en analitzar i determinar qui és el responsable de cada acció. No a nivell de mètode, això queda a un abast llunyà d'aquesta fase, sinó a nivell de classes. Cal fer una llista dels requisits del sistema i les funcionalitats que haurà d'agregar. A partir d'aquesta llista es van determinant els components, i cada funcionalitat serà afegida al component que tingui la responsabilitat de realitzar-la.

Per fer-ho s'han utilitzat els patrons GRASP d'assignació de responsabilitats.

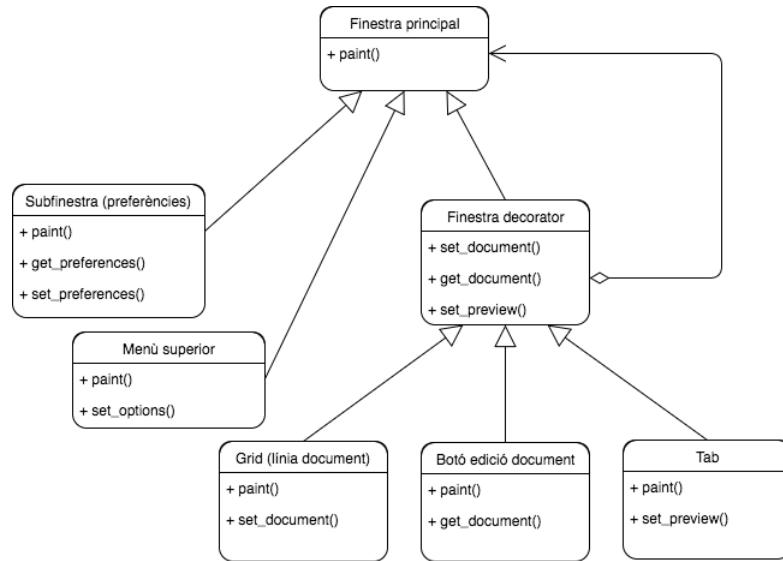
### **8.11.1 Patró de responsabilitats base**

El següent diagrama mostra les responsabilitats bàsiques de l'aplicació:



### **8.11.2 Patró de responsabilitats de la interfície gràfica**

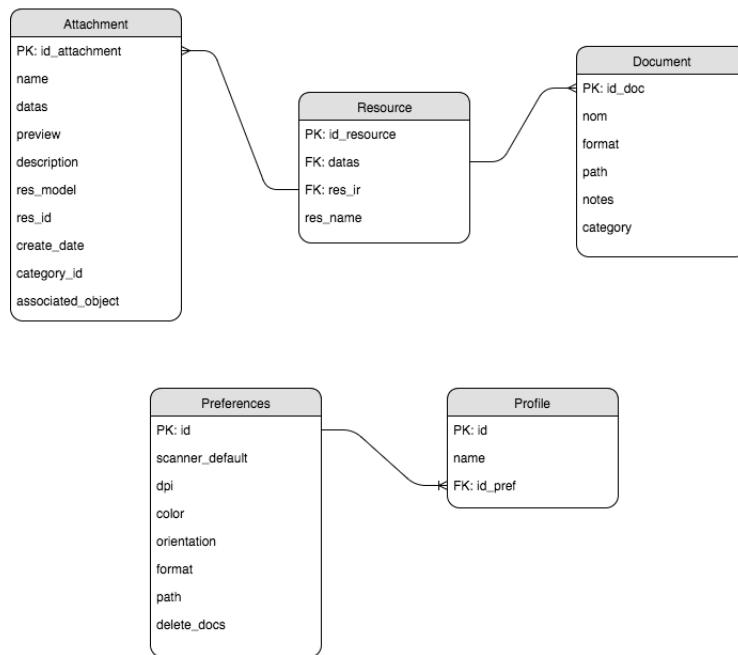
El següent diagrama mostra les responsabilitats bàsiques de la interfície gràfica:



## 8.12 Models de dades

Com que els documents s'adjunten i classifiquen directament al recurs de l'ERP des d'on s'hagi iniciat l'ScannerApp, per definir el diagrama de models de dades, com que no sabem quin és, es pressuposa un recurs amb els paràmetres més típics de la majoria de models de l'ERP. Cal apuntar també que la taula documents no es modela a la base de dades, ja que es guarden les dades per aquella sessió, i després no calen més perquè es crea l'attachment i es classifica i adjunta a l'ERP, on en aquest sistema si que es modela el document (taula attachment).

A continuació es mostren els diagrames E/R per documents i preferències:



Els documents queden persistents a la base de dades mongoDB del servidor de l'ERP, i les preferències

queden persistents a la base de dades SQLite de l'ordinador de cada client. Els perfils no es comparteixen, a diferència dels documents.



# CAPÍTOL 9

---

## Implementació i proves

---

Aquest capítol tracta els problemes apareguts durant el procés d'implementació així com les respectives solucions. També els tests realitzats per tal de certificar que les funcionalitats que es van afegint al software estiguin operatives.

### 9.1 Adaptació sockets natius de Python amb sockets ZMQ

La necessitat d'haver d'usar comunicació ZMQ va fer que es realitzés un estudi de la utilització de la llibreria ZMQ per Python2.5. Després de documentar-se al respecte i fer un ànalisis, i primeres proves, va sorgir un problema. El problema és el següent:

Python2.5 support is deprecated

La solució proposada va ser, intentar l'adaptació de sockets ZMQ als natius de Python. És una bona solució, però poden sorgir alguns problemes. Recordem primer quina és la metodologia de treball dels sockets en la majoria de llenguatges com C, C++, Java...

Server

```
HOST = "X"
PORT = int
servidor.bind((HOST, PORT))
client = servidor.accept() #es queda esperant una connexió
```

Client

```
HOST = "X"
PORT = int
client.connect((HOST, PORT))
```

Finalment quan el client fa la petició de connect, el servidor rep un token del client, que pot ser servir per enviar o rebre:

```
SERVER
-----
client.send()
recv = client.recv()
#pot manejar més conexions amb client2 = servidor.accept()
```

Notar que el servidor necessita un token del client per enviar resposta. A ZMQ en la majoria d'arquitectures no passa això. Posem-ne un exemple: volem disposar de un Client i un Servidor, tan sols un. Per tant utilitzarem REQ, REP, i ni el client ni el servidor necessitarán tokens, ja que estaràn orientats a la connexió i s'hi connectarà quan faci el connect, ningú es quedará en accept. El problema surgeix quan amb un ZMQ STREAM es vol enviar a un socket natiu que si que espera un recv en concret amb un token en concret. La solució passa en desencapsular el primer missatge que rep el server, ja que és del tipus

```
[client_id, message]
```

Amb aquest client\_id es pot fer que el servidor envii missatges tipus multipart especificant el token del client (client\_id) i incloent-hi el missatge

## 9.2 Twain

L'ús d'aquesta llibreria, ha donat diversos errors relacionats amb compatibilitats entre Sistemes de 32 bits i 64 bits, controladors d'escànners amb certes versions per diferents arquitectures... El qué i el com d'aquestes fallades es detallen més extensament en capítols posteriors referents a la implantació, ja que cronològicament amb la definició i explicació del desenvolupament seran més acords amb l'esquema d'aquest document.

## 9.3 Visors PDF

S'havia d'implementar una vista per previsualitzar PDFs generats/obtinguts amb l'ScannerApp. Potser pot semblar un aspecte trivial, però ha portat més maldecaps de l'ho esperat. Per situar el lector en l'entorn, en el moment de desenvolupar aquesta funcionalitat, ja estava implementada la previsualització d'imatges. S'utilitza una llibreria anomenada `Pillow`, pel tractament d'imatges. Però un PDF és un altre món completament diferent. La previsualització de tots els documents ha d'estar integrada completament a la vista de l'ScannerApp, i donar la possibilitat d'obrirlos amb el programa definit per defecte al sistema. Aquest últim és relativament senzill ja que s'utilitza una crida nativa del sistema per obrir el document, sigui una imatge, un pdf, un fitxer d'audio...

El PDF per tant havia d'integrar-se dins la vista, que recordem està programada amb PyQt. PyQt no té cap widget preparat per fer això, però si que en proposa solucions. Es plantejen dos problemes:

1. Visor poc elegant
2. Falta de suport de la llibreria

No és massa bo desenvolupar una tasca pel simple fet de que funcioni, per tant a l'hora d'integrar nous components a l'aplicació, sempre s'intenta fer un estudi previ dels avantatges i els inconvenients que poden

sorgir. Intentant seguir les indicacions que proposa la pàgina oficial de PyQt, ens adonem que el visor que proposen és poc funcional, és com una vista repintada dins un component natiu de la pròpia vista. A més a més, intentant trobar la solució amb la llibreria proposada, es va detectar un error en el software causat per una dependència. En aquest punt es va analitzar el repositori original i comprovar si hi havia alguna Issue oberta sobre aquest tema. N'hi havia una, però ningú es va dedicar a resoldre-la, o a respondre amb una solució. Òbviament quan es detecta això, cal fugir i evitar conflictes en un futur. Per tant es va decidir buscar una altra solució.

Enllaç de la Issue: <https://github.com/wbsoft/python-poppler-qt5/issues/14>

PyQt incorpora un paquet anomenat `QtWebEngineWidgets`, que com el seu nom introduceix, s'utilitza per treballar amb aspectes o característiques web. Si es prova d'obrir un fitxer PDF en un navegador mitjanament modern, veiem que es previsualitza correctament. La solució al problema implicava crear un navegador web integrat a la vista i previsualitzar-ne el document a dins. Era una solució molt més elegant i funcional que el que proposa directament PyQt. Aquest navegador web es genera i s'inclou a la vista de la següent manera:

```
layout = QVBoxLayout()

web = QtWebEngineWidgets.QWebEngineView()
web.settings().setAttribute(QtWebEngineWidgets.QWebEngineSettings.
    PluginsEnabled, True)
web.load(QtCore.QUrl('file:///path_to_file/test.html')) # local url
web.show()

layout.addWidget(web)
```

Interessa obrir una URL local en comptes de fer una petició en una pàgina web, és per això que s'utilitza el paràmetre `file:///`. Però el navegadro que es genera amb PyQt, no és un navegador modern, no té Javascript, no té la majoria de funcionalitats que tenen els navegadors actuals, i per tant no té un renderitzador de pdfs. Per tant a l'executar això, no fallava res, però senzillament no es previsualitzava el PDF. Existeix una solució proposada per l'empresa Mozilla, utilitzada per analitzar i renderitzar PDFs en un navegador web estàndard, desenvolupada amb Javascript. <https://mozilla.github.io/pdf.js/>

Es pot descarregar una versió estable de l'enllaç anterior, o descarregar-se el codi font:

```
git clone git://github.com/mozilla/pdf.js.git
cd pdf.js
```

Aquesta eina que porta el nom de `pdf.js`, consta de una capa de display: `pdf.js` i una capa de nucli `pdf.worker.js`, que renderitzen en un visor web predefinit un fitxer PDF donat (capa de vista).

Per utilitzar aquesta eina, sols s'ha de passar a la capa vista el fitxer PDF, i sense executar res més, a l'utilitzar aquest PDF, la capa de vista agrega les funcionalitats Javascript per renderitzar-lo.

Però `pdf.js` té moltes més versions, i aquesta funcionalitat és la bàsica, i no quedava un visor elegant, ja que simplement renderitza el pdf sense agregar una capçalera de vista, o una funcionalitat per ampliar el document...

Aquest punt va ser un autèntic maldecap, perquè subjectivament no dónen un bon suport, i no hi ha una bona documentació per entendre com funcionen les altres característiques de `pdf.js`. I simplement es volia agregar una capçalera al visor.

Així que va tocar entendre com funcionaven les altres característiques, i desenvolupar amb Javascript una funcionalitat per poder agregar el visor. La capçalera agrega un component a la capa de vista viewer.js. No va ser trivial agregar aquesta funcionalitat, perquè amb aquesta nova capa no n'hi havia prou passant el PDF, a més es va haver d'analitzar com funcionava i proposar-ne una solució, ja que aquesta capa espera un document del tipus binari per crear una instància de Uint8Array. La solució va ser codificar el document en base64, decodificar-lo per passar-lo a una classe que ho converteix al tipus Uint8Array i utilitzar la funció PDFViewerApplication.open(content) per renderitzar el document PDF. es van crear varis fitxers per guardar i parsejar el document en base 64, codificar i decodificar el document, com segueix:

render.py

```
import base64

class PDFRender(object):
    def __init__(self, pdf_file):
        self tmpl = '../pdfjs/web/open_b64.js'
        self.pdf_file = pdf_file

    def render(self, output):
        try:
            with open(self tmpl, 'r') as f:
                tmpl = f.read()
            with open(self.pdf_file, 'rb') as f:
                b64 = base64.b64encode(f.read())
            with open(output, 'w') as f:
                f.write(tmpl % b64.decode('utf-8')) # b'' to str()
        except Exception as err:
            print(err)

if __name__ == '__main__':
    render = PDFRender('proyecto.pdf')
    render.render('open_b64.js')
```

open\_b64.js

```
function open_b64() {
    var bin = atob('%s');
    var content = new Uint8Array(bin.length);
    for (var i = 0; i < bin.length; i++) {
        content[i] = bin.charCodeAt(i);
    }
    PDFViewerApplication.open(content);
}

if (document.readyState === 'interactive' || document.readyState === 'complete
→') {
    open_b64();
} else {
    document.addEventListener('DOMContentLoaded', open_b64, true);
```

A més hi havia una inconsistència per utilitzar-lo en el nostre cas. Aquesta capçalera integra moltes funcions i realment és molt còmode i interessant d'utilitzar, però una de les funcions és carregar un PDF. Òbviament si s'està treballant des de l'ScannerApp, i es visualitza un document, s'està visualitzant *aquest* document, i

per tant no es pot carregar cap altre, sino es perd accés al primer, i es torna inconsistent, ja que s'ha obtingut un document A i s'hi carrega un document B. Per desactivar aquesta opció es va fer un estudi de la capa de vista `view.js` i es va desactivar aquesta funcionalitat:

```
appConfig.toolbar.openFile.setAttribute('hidden', 'true');
appConfig.secondaryToolbar.openFileButton.setAttribute('hidden', 'true');
```

## 9.4 Instal·lador

Existeixen diverses solucions per crear un executable o un instal·lador d'un programa per Windows amb Python. En capítols anteriors s'han donat detalls de la llibreria escollida per crear l'instal·lador que es necessita en aquest projecte `cx_Freeze`. N'hi ha d'altres com `py2exe`, però s'ha quedat enrere en comparativa a la utilitzada. A l'hora de desenvolupar cal tenir molt en compte que les coses no ocorren igual, i s'han detectat algunes funcionalitats no ben complertes a l'hora de desplegar-ho en la màquina del client. Un dels problemes apareixia a l'executar el programa instal·lat a la màquina del client, ja que demanava executar-se sempre a mode administrador. Provenia ja que a l'hora de fer la creació de la base de dades amb l'instal·lador a la màquina del client, s'havia configurat el setup de `cx_Freeze` de tal manera realitzés una importació de la base de dades a una ruta en concret on demanava permisos d'administrador per executar-la. Es va solucionar situant-la a una ruta del perfil de l'usuari.

També ha calgut un estudi exhaustiu a l'hora d'importar algunes de les dependències necessàries pel que respecte a les llibreries requerides pel programa. Per incloure les llibreries en la versió instal·lable, cal crear un diccionari de diccionaris `options`, i afegir-hi una llista de `packages`, que conté els paquets necessaris.

```
options = {
    'build_exe': {
        'packages': [
            'PyQt5.uic',
            'PyQt5.QtGui',
            'PyQt5.QtWidgets',
            'PyQt5.QtCore',
            'PIL',
            'zmq',
            'os',
            'tkinter',
            'raven',
        ]
    }
}
```

Un dels problemes residia en la llibreria PIL, que s'utilitzava el paquet Image de la mateixa. Per no carregar l'instal·lador de Mb innecessaris s'intenten fer imports relatius als paquets que s'utilitzen, i no de tota la llibreria. Per tant s'utilitzava l'import de `PIL.Image`. Però quan s'instal·lava el programa no funcionava correctament una funcionalitat de PIL relacionada amb els PDF, en canvi en la versió no compilada, funcionava sense problemes. Es va obrir un tema a StackOverflow: <https://stackoverflow.com/questions/43804832/python-image-library-cant-convert-to-pdf>, però finalment sense aquesta ajuda es va trobar la solució. El problema provenia de que la llibreria utilitzava un paquet pilpdf que no estava inclòs en el paquet Image, però la versió no compilada tenia coneixença del source de PIL i sabia trobar aquest paquet, en canvi

la versió instal·lada no el tenia i per tant importava i copiava només el paquet Image, per tant no es troava. Això es va detectar fent un estudi d'aquesta llibreria i es va solucionar substituïnt: PIL. Image per PIL.

## 9.5 Accés directe del software automàticament

S'havia parlat amb el client, que era interessant incloure un accés directe del programa a l'escriptori de forma automàtica, just després de finalitzar el procés d'instal·lació. cx\_Freeze proposa utilitzar el paràmetre `shortcutDir`, font: <http://cx-freeze.readthedocs.io/en/latest/distutils.html>, però al realitzar la instal·lació, es crea l'accés directe però no té el paràmetre *Iniciar en*, per tant és com una còpia de l'accés directe, i no té el path d'on s'han d'executar els recursos, cosa que provocava un mal funcionament. Per solucionar això cal donar la taula d'accisos de la següent forma:

```
shortcut_table = [
    ("DesktopShortcut",           # Shortcut
     "DesktopFolder",            # Directory_
     "ScannerApp Connector",    # Name
     "TARGETDIR",                # Component_
     "[TARGETDIR]connector.exe", # Target
     None,                      # Arguments
     None,                      # Description
     None,                      # Hotkey
     None,                      # Icon
     None,                      # IconIndex
     None,                      # ShowCmd
     'TARGETDIR'                 # WkDir
   )
]

# Crear taula d'accisos
msi_data = {'Shortcut': shortcut_table}

# Canviar opçons per defecte MSI i especificar la taula definida
bdist_msi_options = {'data': msi_data}
```

## 9.6 Compiled Version vs Uncompiled Version

Un cop instal·lada la versió previament compilada, les coses canvien versus a la versió no compilada que s'utilitza per desenvolupar i treballar in-situ. A l'hora d'executar el programa amb la versió instal·lada canvien algunes coses: canvia el path per exemple, tot allò que es presuposa que s'executa des d'un cert directori haurà canviat, per exemple, Windows instal·la els programes per defecte a C:\Program Files. Les dependències, si el software utilitza una certa llibreria, també l'ha de saber trobar des del directori d'instal·lació, i un llarg etc. de característiques que són necessàries tenir clares. Per tant cal entendre bé el que fa la llibreria a l'hora de crear l'instal·lador i conèixer les característiques internes del llenguatge, en aquest cas Python.

Es necessitava executar des de un script de Python, un altre script de Python, i en la versió instal·lada no funcionava. Tampoc funcionaven certes llibreries.

Per treure l'entrellat del problema, com es comenta primer s'ha d'entendre el següent:

Python és un llenguatge interpretat, que s'executa utilitzant un programa intèrpret, enlloc de compilar el codi a llenguatge màquina que es pugui comprendre i executar directament a la màquina. Aquests tipus de llenguatge són molt ràpids, i el que realment fa l'intèrpret de Python és traduir el codi font a un pseudocodi màquina, el bytecode en arxius .pyc.

cx\_Freeze que és la llibreria per crear l'instal·lador, realitza un paquet en un directori copiant el source de les llibreries marcades com a dependències, interpreta el programa i genera els arxius en bytecode .pyc, incloent-los també en el directori, i per últim genera els .exe marcats en el fitxer de setup com a executables, i també els inclou dins el directori. Quedant una estructura d'arbre totalment diferent de la de la versió no compilada.

Òbviament canvien molts les coses, i es va haver d'analitzar com treballava aquesta llibreria i buscar una solució per diferenciar una versió de l'altre per actuar en conseqüència. Un altre detall és que en aquesta versió no s'executen els fitxers .py, sinó que ho han de fer els .exe. Per tant calia un mètode per esbrinar quina versió estava corrent per muntar una estructura o un altre, o per executar uns fitxers o uns altres.

Es van proposar dues coses, la primera és un mètode per detectar el tipus de versió:

```
if getattr(sys, 'frozen', False):
    # frozen version
else:
    # unfrozen version
```

I la segona, amb intenció de generar un codi funcional i reutilitzable, és crear un path general com a constant SCAN\_PATH. Per fer-ho la llibreria os inclosa com a paquet bàsic de Python, incorpora la funcionalitat os.getcwd(), que retorna en format string el path actual de treball. Cal anotar que al canviar l'estructura, on abans ens cercaven els fitxers, ara haurà canviat degut a l'estructura que ha muntat cx\_Freeze. Per tant haurem de traçar camins diferents. D'aquesta manera es pot distingir entre versions i anar a buscar les coses on són, i actuar en conseqüència. Anteriorment s'ha parlat de que un procés Python necessitava executar-ne un altre, a continuació es mostra com s'ha solucionat això per poder fer-ho amb les dues versions:

```
import subprocess
if getattr(sys, 'frozen', False):
    # frozen
    path = r'{0}{1}'.format(SCAN_PATH, '/GUI.exe')
    self=subprocess.Popen(path)
else:
    # unfrozen
    path = r'{0} {1}{2}'.format('python', SCAN_PATH.rsplit('agent')[0],
                                'GUI/GUI.py')
    self=subprocess.Popen(path)
```

Notem que en el primer cas s'executa el .exe, i en el segon el .py. Aquesta solució no només ha servit per aquest tipus de fitxers, sinó per molts altres temes, com els fitxers de vista. Recordar que són fitxers .ui amb la definició xml del disseny de la interfície d'usuari.

## **9.7 Navegació ERP**

Per iniciar l'ScannerApp cal estar dins un element de l'ERP, ja que un cop acabada la feina del programa, de forma intel·ligent buscarà la o les id's d'aquest element per classificar-hi els documents adjunts, i pujar-los a l'ERP. No té iniciar l'ScannerApp des de cap objecte, perquè no s'està indicant la classificació necessària. Per tal de preveure errors, s'ha proposat llançar un prompt a l'usuari i no permetre l'inici del programa si no s'està a cap element de l'ERP:

```
import common  
common.warning('You must resource a object', 'Warning')
```

## Implantació i resultats

---

Per descriure el treball desenvolupat per implantar el sistema, es divideix la documentació següent en diferents apartats:

### 10.1 Integració Sentry i Logger

En el capítol de requisits, s'exposa l'integració de l'aplicació amb dos sistemes, sentry i logger, pel monitoratge i control d'errors. A partir d'aquest punt del projecte serà habitual veure talls de codi on s'utilitza la captura de certs events. Per estar més familiaritzats, a continuació s'exposa com s'ha integrat l'aplicació amb Sentry i Logger.

A la classe principal on es vulgui capturar events, s'integrarà el Sentry i el Logger. Les classes instanciades, derivades, heredades... d'aquestes principals es podran subscriure als missatges i utilitzar-ne el mateix sistema de captura d'events o errors. Per tant només caldrà integrar aquests dos sistemes a les classes principals de cada procés, en l'àmbit d'aquest treball (connector i client).

```
def setup_log():
    logger = logging.getLogger('connector')
    formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s -
→%(message)s')
    user_path = r'{0}/{1}'.format(environ['USERPROFILE'], 'logs.log')
    hdlr = logging.FileHandler(user_path)
    hdlr.setFormatter(formatter)
    logger.addHandler(hdlr)
    logger.setLevel(logging.INFO)

    sentry = Client('http://id@sentry.gisce.net/103')
    logs = (sentry, logger)

    return logs
```

```
if __name__ == '__main__':
    sentry_client, logger = setup_log()
    app = NameClass(sentry_client, logger)
    app.main()
```

## 10.2 Sockets configurables

Per la configuració de @IP, #Port i Protocol dels sockets, s'ha utilitzat una llibreria anomenada configparser. Que obté informació d'un o diversos fitxers de configuració. Això és molt còmode, ja que si es necessiten canviar aquestes configuracions, ja que només s'ha de canviar l'arxiu de configuració, sense haver de tocar el codi font. O quan s'hagi de desplegar a diversos clients, cadascun podrà tenir el seu fitxer de configuració. També presenta un codi més elegant, ja que no és massa bo deixar algunes coses en *hardcode*. El fitxer de configuració utilitzat du el nom de config.cfg. En el projecte s'ha utilitzat el configparser de la següent manera:

```
import configparser

parser = configparser.ConfigParser()
parser.read('config.cfg')
port = parser.getint('PORT', 'port')
ip = parser.items('HOST')[0][1]
protocol = parser.items('PROTOCOL')[0][1]
```

## 10.3 GUI

A continuació es defineix la implementació de una vista d'usuari, per gestionar i interactuar amb l'ScannerApp.

Atés a la guia de disseny de l'aplicació, es necessita crear una interfície d'usuari independent del core de l'aplicació, per tal de poder utilitzar una altre vista si es necessita o simplement utilitzar les funcionalitats de l'aplicació sense una interfície. Per fer-ho s'ha implementat un client d'aplicació, el qual la interfície o la no interfície utilitzarà per fer peticions de manera estàndard.

Per importar el client es fa de la següent manera:

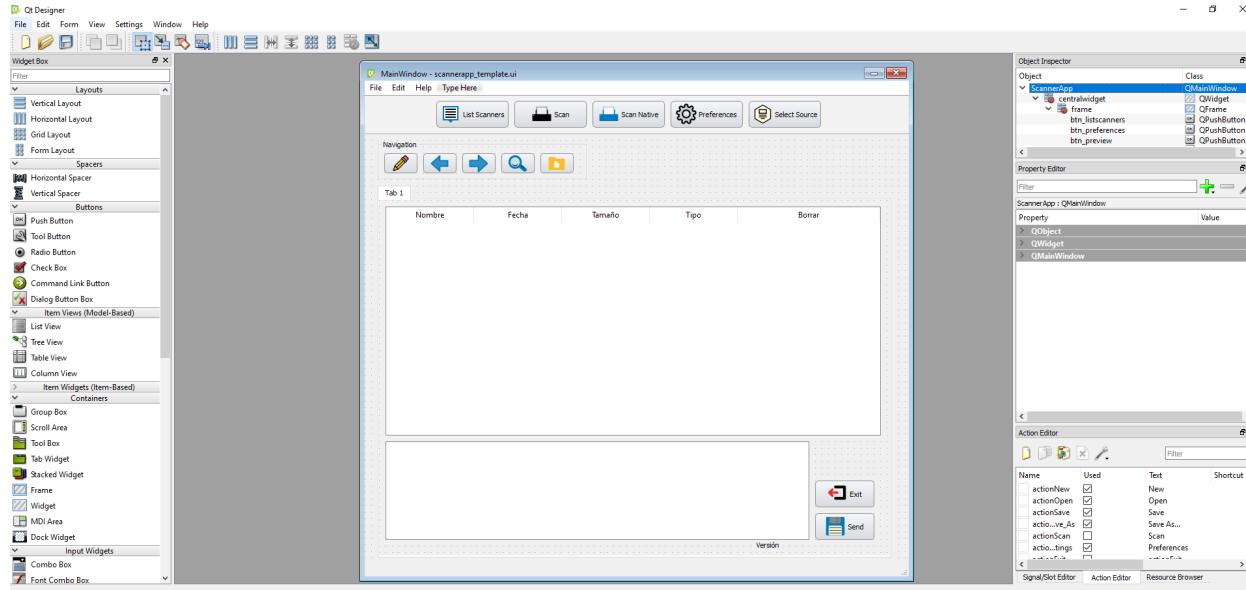
```
from scannerapp.GUI.client import Client
client = Client()
```

Cal instanciar el client per tal de que inici una comunicació amb el nucli del software.

En capítols posteriors es venia comentant que és interessant abstreure al programador del disseny de l'aplicació en el que la programació respecte. És a dir, que hi han eines que estalvién fer un disseny, posicionat, maquetació... de tota l'interfície, i donen pas al que realment ha d'importar més: les funcionalitats de cada element d'aquest disseny. Amb això no s'intenta fer que el programador no hagi de dissenyar l'interfície d'usuari, sinó que pugui dissenyar-la d'una manera més còmode i guanyar temps per centrar-se en l'implementació.

QtDesigner és una eina que permet dibuixar una interfície d'usuari i generar-ne un fitxer de vista amb xml .ui.

S'ha utilitzat aquesta eina per dibuixar l'interfície d'usuari, excepte alguns casos que es tractarà més endavant que han obligat a dibuixar-los i maquetarlos via codi font.



Per utilitzar aquests fitxers de vista amb PyQt, s'ha de fer de la següent manera:

```
class ScanGui(QMainWindow):
    def __init__(self):
        super(ScanGui, self).__init__()
        uic.loadUi(_template, self)
        self.show()
```

### 10.3.1 Vista principal

A l'iniciar la vista, es carrega el fitxer de vista com es mostra en l'exemple anterior per tenir accés a tots els elements de la vista, i posteriorment s'estableix l'estat inicial de l'aplicació. En el nostre cas comptem amb diversos elements a la vista: botons, pestanyes, containers... Generalment quan parlem d'elements de la vista utilitzarem el terme widgets.

Si s'hagués de descriure cada línia de codi, s'ampliaria molt aquesta memòria. Per tant a continuació es mostra el mètode principal i seguidament se'n comenten les operacions més rellevants.

```
def init_ui(self):
    self.setWindowIcon(QtGui.QIcon('{0}\{1}'.format(_images, 'scan.png')))
    self.btn_openfile.setIcon(QtGui.QPixmap(r'{0}/{1}'.format(_images, 'folder.png')))
    self.btn_openfile.clicked.connect(self.open_image)

    ''' Buttons '''
    self._list_btns = [self.btn_scan, self.btn_preview, self.btn_listscanners,
                       self.btn_scanner]
```

```
for button in self._list_btns:
    type, method = str.split(button.objectName(), '_')
    button.clicked.connect(getattr(self, method))

self.btn_preferences.clicked.connect(self.print_dialog)
self.confirm_button.clicked.connect(self.confirm)
self.exit_button.clicked.connect(self.exit)
self.edit_attachment_button.clicked.connect(lambda: self.cell_double_
→clicked(self.tableWidget.currentItem()))
self.edit_attachment_button.setEnabled(False)

''' Actions '''
self.actionSettings.triggered.connect(self.print_dialog)
self.actionNew.triggered.connect(self.open_image)
self.actionSave.triggered.connect(self.show_file_dialog)
self.actionExit.triggered.connect(self.close_app)

''' Containers '''
self.tableWidget.doubleClicked[QtCore.QModelIndex].connect(self.cell_
→double_clicked)
self.tableWidget.clicked[QtCore.QModelIndex].connect(self.cell_clicked)
self.tableWidget.selectionModel().selectionChanged.connect(self.change_
→state_button)
self.tableWidget.setSizeAdjustPolicy(QAbstractScrollArea.AdjustToContents)

''' Fix resizes '''
self.tableWidget.horizontalHeader().resizeSection(0, 300)
self.tableWidget.horizontalHeader().resizeSection(2, 120)
self.tableWidget.horizontalHeader().resizeSection(3, 120)
self.tableWidget.horizontalHeader().resizeSection(4, 50)
self.tableWidget.horizontalHeader().setSectionResizeMode(QHeaderView.
→Fixed)
self.tableWidget.verticalHeader().setSectionResizeMode(QHeaderView.Fixed)

''' Tabs '''
self.tabWidget.setTabText(0, "Documents")
self.tabWidget.setTabText(1, "Preview")
self.tabWidget.setTabsClosable(True)
self.tabWidget.tabCloseRequested.connect(self.remove_tab)
self.tabWidget.currentChanged.connect(self.change_tab)

'''Window Hint'''
self.setWindowFlags(QtCore.Qt.WindowStaysOnTopHint)
self.show()
self.setWindowFlags(QtCore.Qt.X11BypassWindowManagerHint)
self.show() # You must call show() to make the widget visible again
```

**Botons:** la connexió dels botons principals es fan aprofitant el mètode “getattr()” de Python, que concretament obté un atribut o mètode de classe, tant self com externa. S’aprofita per crear una nomenclatura de botons amb el següent prefix btn\_method. Ex: btn\_scan, btn\_listscanners. I posteriorment connectar-los amb el seu mètode corresponent.

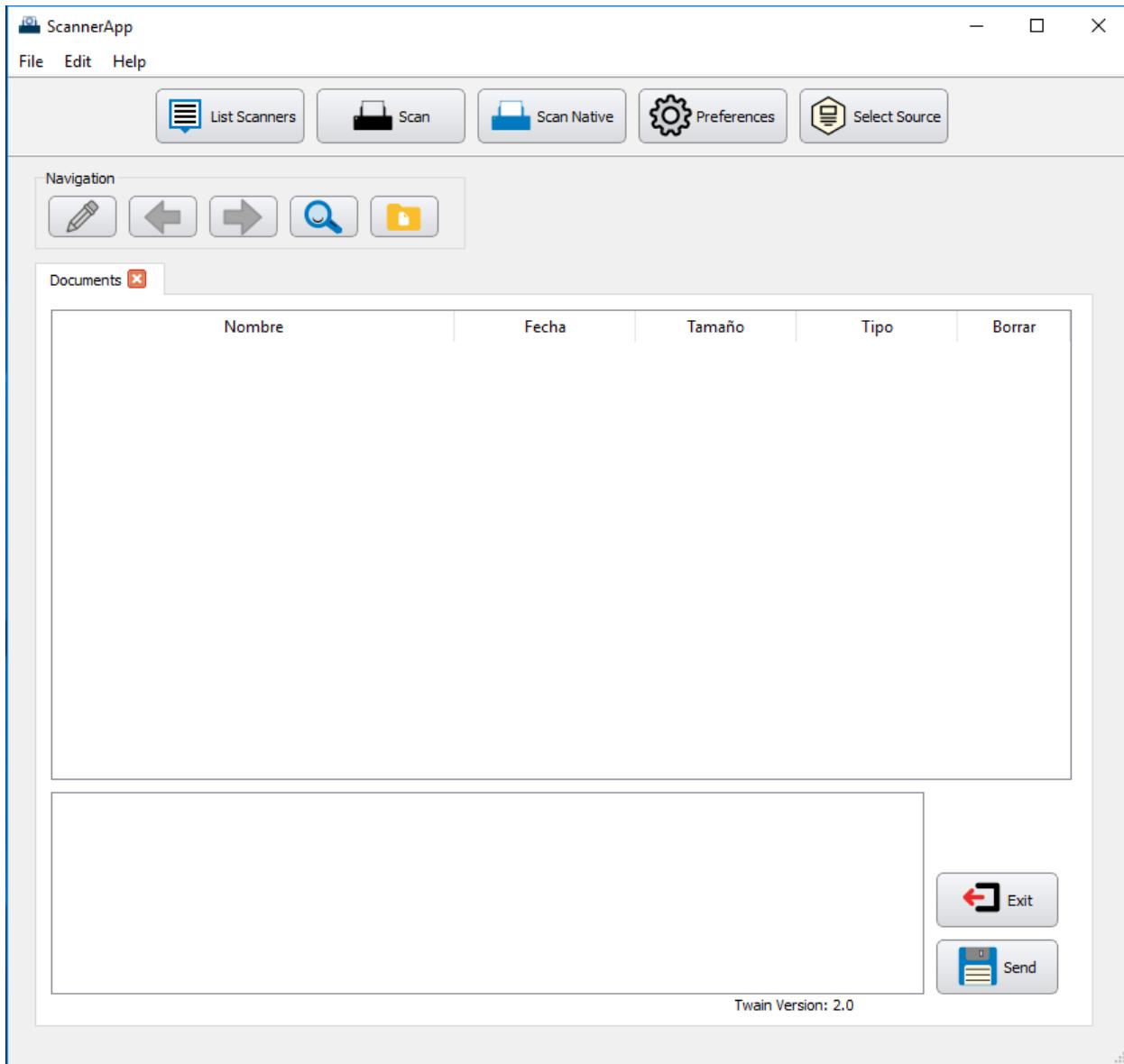
**Senyals:** per esbrinar els senyals que pot emetre cada widget, es pot revisar la documentació de widgets de PyQt: <http://doc.qt.io/qt-5/qtwidgets-module.html>. Per exemple per mostrar una llista dels documents obtinguts s'utilitza un widget anomenat qtablewidget, i ens podem connectar al senyal de doble click per mostrar i previsualitzar el document. (s'utilitza també el mètode botó per donar més funcionalitat a la vista).

**Accions:** tal com es connecta amb un senyal via slot, es poden connectar accions natives de PyQt o accions predefinides. Per exemple, en el nostre cas si l'usuari tanca l'aplicació, no interessa perdre els documents obtinguts, per tant ens connectem a l'acció “actionExit” i realitzem el necessari.

**Pestanyes:** com es mostra en el passat tall de codi la vista incorpora un sistema de pestanyes per mostrar la vista principal dels documents i anar obrint documents en pestanyes per editar-los i visualitzar-los. Menys la pestanya principal, la resta les fem *closables*.

**Finestra principal:** per tal de mostrar la finestra al top de les possibles finestres obertes, es pot utilitzar el mètode “setWindowFlags”

Aquest mètode pinta la vista com es mostra a la imatge que prossegueix:



### Interacció amb el client

La vista actual utilitza les següents funcions del client:

```
self.client.set_preferences({prefkey: value})
self.client.get_preferences()
self.client.list_scanners()
self.client.get_path()
self.client.scan()
self.client.scan_native()
self.client.attach_file(attachment)
self.client.delete_file(attachment)
self.client.protocol_version()
self.client.close()
```

D'aquesta manera l'interacció amb el nucli de l'aplicació és transparent per la vista.

### Sistema de prompts

És interessant utilitzar un sistema de prompts per informar a l'usuari d'un esdeveniment a l'aplicació. Per fer-ho s'ha creat una classe que implementa 4 nivells d'informació: information, warning, critical, question. Aquest últim mostra un combo de butons *Ok*, *Cancel* per demanar confirmació a l'usuari.

```
class Prompt:
    def show(self, message=None, type=None):
        ''' Types: QMessageBox.Question, QMessageBox.Information, QMessageBox.
        →Warning, QMessageBox.Critical '''
        name = 'ScannerApp'
        if not type:
            QMessageBox.information(None, name, message)
        elif type == 'question':
            QMessageBox.question(None, name, message)
        elif type == 'information':
            QMessageBox.information(None, name, message)
        elif type == 'warning':
            QMessageBox.warning(None, name, message)
        elif type == 'critical':
            QMessageBox.critical(None, name, message)
        else:
            QMessageBox.information(None, name, message)

    def question(self, message):
        name = 'ScannerApp'
        button_reply = QMessageBox.question(None, name, message,
                                            QMessageBox.Yes | QMessageBox.No,
                                            QMessageBox.No)

        if button_reply == QMessageBox.Yes:
            return True
        else:
            return False
```

### 10.3.2 Mètodes principals

Després de fer una petició al client perquè obtingui el document, es pinta a la taula, a partir d'ara grid, de la següent manera:

```
def setTableWidgetItems(self):
    _configs = self.client.get_preferences()
    path = self.client.get_path()
    output_format = _configs['format']

    if getattr(sys, 'frozen', False):
        _images = r'{0}{1}'.format(SCAN_PATH+'scannerapp', '/GUI/images')
```

```
else:
    _images = r'{0}{1}'.format(SCAN_PATH.rsplit('agent')[0], '/GUI/
↪images')

_items = list()

from os import stat
try:
    size = stat(path).st_size
except Exception as error:
    self.prompt.show(message=str(error), type="critical")
    raise

size = (size / (1024 ** 2))
date = str(datetime.datetime.now()).split('.')[0]

''' Table Widget '''
row_position = self.tableWidget.rowCount()
self.tableWidget.insertRow(row_position)

_values = [path, date, '{0:.3g} {1}'.format(size, "MB"), str(output_
↪format)]
for value in _values:
    item = QTableWidgetItem(value)
    item.setTextAlignment(QtCore.Qt.AlignCenter)
    _items.append(item)

''' Delete icon '''
item_five = QLabel()
pixmap = QtGui.QPixmap('{0}\{1}'.format(_images, 'delete.png'))
item_five.setPixmap(pixmap)
item_five.setAlignment(QtCore.Qt.AlignCenter)
_items.append(item_five)

for position, item in enumerate(_items):
    if position != 4:
        self.tableWidget.setItem(row_position, position, item)
    else:
        self.tableWidget.setCellWidget(row_position, position, item)

self.tableWidget.setSelectionBehavior(QTableView.SelectRows) # Select_
↪all columns the row
self.tableWidget.setEditTriggers(QAbstractItemView.NoEditTriggers) #_
↪Set not editable
```

Bàsicament obté la informació del document, i el pinta al grid amb el mètode “setItem” o “setCellWidget” (aquest últim en cas de voler pintar una icona a una cel·la de la taula).

El mètode següent, envia els documents que té al grid al client perquè els adjungi, mostra una barra de progrés per donar un feedback a l'usuari i elimina els documents del disc si està activada l'opció d'eliminar arxius en sortir de l'aplicació:

```
def attach_process(self):
    num_attachments = self.tableWidget.rowCount()
    progress = QProgressDialog("Attaching...", "Abort Attach", 0, num_attachments, self)
    progress.setWindowModality(QtCore.Qt.WindowModal)
    progress.setWindowTitle("Attachments")
    progress.show()
    delete_files = self.get_preferences()['delete_files']

    for row in range(self.tableWidget.rowCount()):
        QApplication.processEvents()
        progress.setValue(row)
        attachment = self.tableWidget.item(row, 0).text()
        self.client.attach_file(attachment)
        time.sleep(1)
        if delete_files:
            remove(attachment)

    if progress.wasCanceled():
        break

    progress.setValue(num_attachments)
```

Quan es vulgui visualitzar un document, es crea una pestanya si és que no està oberta, procurant el tipus de visualització. Atès que anteriorment s'ha mencionat que la majoria d'elements de la vista eren dissenyats amb “QtDesigner”, vegem-ne un exemple de com crear-ne un des de codi. En aquests casos és bastant còmode tot i que a voltes una mica molest utilitzar *layouts*. A més de la lògica típica de cada widget, també s'afegeix una màscara per validar noms correctes o incorrectes de documents, procés d'edició...

Previsualització a mode pestanya d'un BMP:

```
pix = QtGui.QPixmap(attachment)
self.tab = QWidget()
self.tabWidget.addTab(self.tab, attachment_name)
self.tab.setAccessibleName(attachment)

''' Layouts '''
main_layout = QHBoxLayout()
data_layout = QFormLayout()
image_layout = QVBoxLayout()
confirm_layout = QHBoxLayout()

''' Labels '''
name_label = QLabel("Nombre de archivo")
category_label = QLabel("Categoría")
notes_label = QLabel("Notas")

''' Inputs '''
name_input = QLineEdit(attachment_name.split('.')[0])
name_input.setMaxLength(30)
category_input = QLineEdit()
notes_input = QTextEdit()
image = QLabel()
```

```
image.setPixmap(pix.scaled(650, 650, QtCore.Qt.KeepAspectRatio))
image.setMidLineWidth(2000)

''' Validator '''
regexp = QtCore.QRegExp('^[A-Za-z0-9_]+$', 0, QtCore.QRegExp.CaseInsensitive)
validator = QtGui.QRegExpValidator(regexp)
name_input.setValidator(validator)

''' Buttons '''
_images = r'{0}{1}'.format(SCAN_PATH.rsplit('agent')[0], '/GUI/images')
confirm = QPushButton("Save")
open = QPushButton("Open")
open.setIcon(QtGui.QIcon('{0}\{1}'.format(_images, 'folder.png')))
confirm.setIcon(QtGui.QIcon('{0}\{1}'.format(_images, 'save.png')))

image_layout.addWidget(image)
data_layout.addWidget(name_label)
data_layout.addWidget(name_input)
data_layout.addWidget(category_label)
data_layout.addWidget(category_input)
data_layout.addWidget(notes_label)
data_layout.addWidget(notes_input)
confirm_layout.addWidget(open)
confirm_layout.addWidget(confirm)

data_layout.addItem(confirm_layout)

main_layout.addLayout(image_layout)
main_layout.addLayout(data_layout)

self.tab.setLayout(main_layout)
self.tabWidget.setCurrentWidget(self.tab)

''' Params: index, row, data_new, data_old '''
confirm.clicked.connect(lambda: self.close_tab(self.tabWidget.currentIndex(),
                                             index.row(), r'{0}/{1}.{2}'.format(attachment.rsplit('/', 1)[0], name_input.
                                             text(), format), attachment))
open.clicked.connect(lambda: self.open_imagee(attachment))
```

Eliminació de documents:

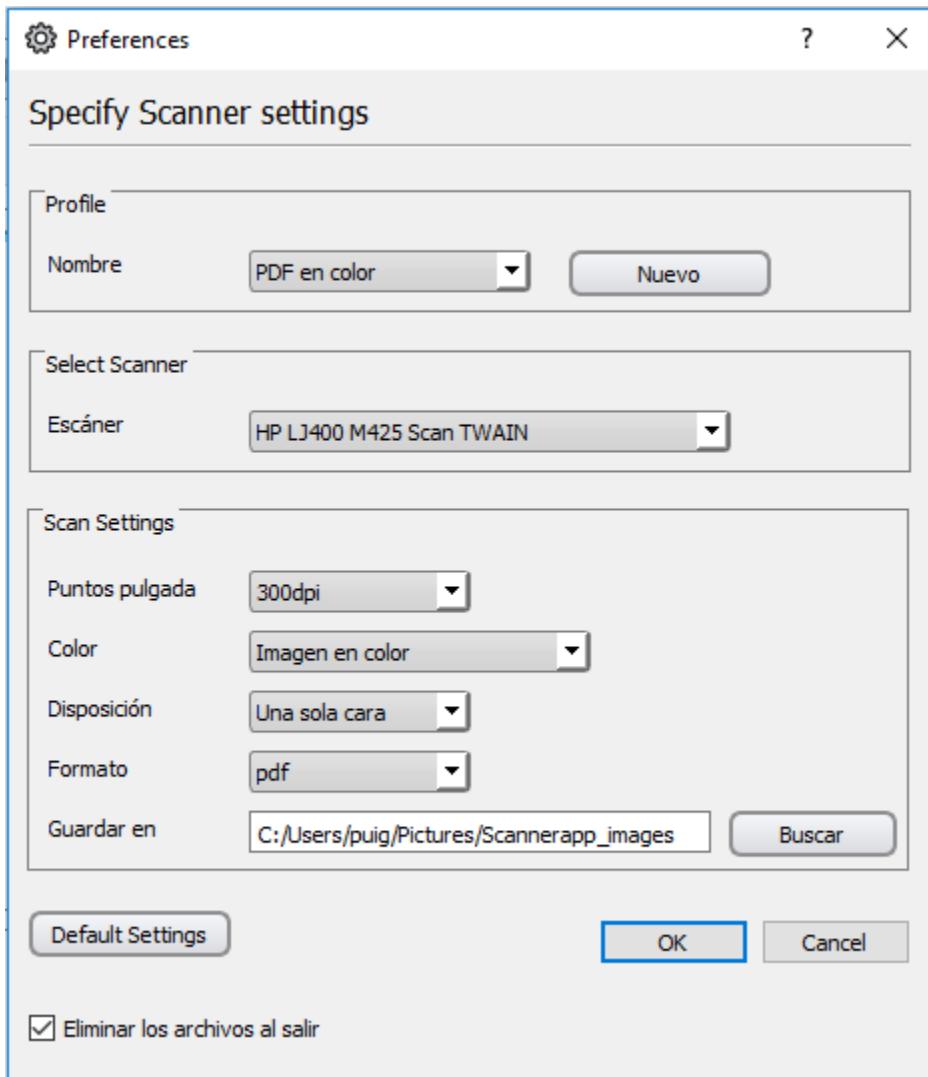
```
def cell_clicked(self, index):
    if index.column() == 4:
        _exec = self.prompt.question(message='Are you sure?')
        if _exec:
            try:
                # col 0: attachment_name
                _attach = self.tableWidget.item(index.row(), 0).text()
                close_state = self.tab_is_open(_attach)
                if close_state:
                    self.remove_tab(close_state)
```

```
remove(_attach)
self.tableWidget.removeRow(index.row())
self.client.delete_file(str(_attach))
except Exception:
    self.prompt.show(message='Error during deletion',
                     type='critical')
```

### 10.3.3 Vistes secundàries

Una vista secundària es pinta a davant de la vista principal. Concretament s'utilitza una finestra del tipus QDialog.

Es detallarà l'apartat de la vista de preferències.



Com que la forma d'interactuar amb els widgets ja s'ha vist en l'apartat anterior, ens centrarem en la creació d'una subvista:

```

class SubWindow(QDialog):
    def __init__(self, _list_default_scanners, path=None, preferences=None):
        QDialog.__init__(self)

        if getattr(sys, 'frozen', False):
            _template = r'{0}{1}'.format(SCAN_PATH+'/scannerapp', '/GUI/
→templates/dialog_preferences_project.ui')
        else:
            _template = r'{0}{1}'.format(SCAN_PATH.rsplit('agent')[0], '/GUI/
→templates/dialog_preferences_project.ui')

        ''' carreguem el fitxer dialog '''
        uic.loadUi(_template, self)
        self.setWindowTitle('Preferences')
        self.setWindowIcon(QtGui.QIcon(r'{0}{1}'.format(SCAN_PATH.rsplit(
→'agent')[0], '/GUI/images/preferences.png')))
        self.initSubWindow(_list_default_scanners, path, preferences)

```

A continuació es detalla la creació i mètodes de la vista de preferències. Cal apuntar que aquesta vista necessita informació actualitzada pel que refereix l'estat de l'aplicació i de la base de dades. Es necessita saber les preferències actuals de la base de dades, obtenir dades del protocol d'interacció amb els escàners... Concretament ha d'implementar les funcionalitats de:

- Obtenir els escàners disponibles (list\_scanners) i pintar-los a un comboBox.
- Obtenir els perfils disponibles.
- Recuperar/Canviar les preferències de base de dades. Per tal de realitzar un software òptim, la gravació de les preferències implica un canvi a la base de dades, però no s'actualitzen fins que l'usuari confirma el canvi. A més, si la única manera de canviar les preferències és des d'aquesta vista, no caldrà consultar la base de dades cada cop que s'obri aquesta mateixa, ja que l'estat de cada comboBox tindrà la mateixa forma que la base de dades, perquè és l'únic mòdul que la modifica. Per tant només caldrà fer una consulta a l'inici de l'aplicació, i els *inserts* o *updates* cada cop que l'usuari canviï les preferències.
- Recuperar l'estat anterior de la vista. També cal retornar a un estat anterior de la vista. És a dir si l'usuari canvia els estats dels widgets (comboBox i Linedit), i finalment es desdiu de l'opció i cancel·la el procés (botó *cancel*).
- Reiniciar les preferències per defecte Cal canviar a un estat per defecte els widgets i actualitzar-ho a la base de dades si es confirma el canvi.

Per realitzar això s'utilitza un sistema d'estats del widget: actual, antic i confirmat. Per consultar les dades, la finestra principal passa per paràmetre a aquesta classe la informació demandada al client.

A continuació es mostren els algorismes principals:

```

def save_state(self):
    '''guarda el combobox i el seu estat actual per si es
necessita cancel·lar els canvis'''
    _state = dict()
    for elem in self.__dict__.keys():
        if re.match('comboBox*', elem):

```

```

        _state[elem] = getattr(self, elem).currentIndex()
self._path_aux = self._path
self._deletefiles_state = self.checkBox_deletefiles.isChecked()

def cancel(self):
    '''retorna els comboBox al seu estat anterior'''
    for comboBox in self.state:
        setattr(self, comboBox).setCurrentIndex(self.state[comboBox])
    self.textEdit_path.setText(self._path_aux)
    self.checkBox_deletefiles.setChecked(self._deletefiles_state)

def deleteIndex(self, index):
    for elem in self.__dict__.keys():
        if re.match("comboBox*", elem):
            setattr(self, elem).setCurrentIndex(0)
        elif re.match("textEdit_path*", elem):
            setattr(self, elem).clear()

    self.checkBox_deletefiles.setChecked(False)

def savePreferences(self):
    return self.preferences

```

Per guardar l'estat d'un widget:

```

self.comboBox_dpi.activated[str].connect(self.onActivatedDPI)
...
if preferences['dpi']:
    index = self.comboBox_dpi.findText('{0}{1}'.format(preferences['dpi'],
    ↴'dpi'))
    self.comboBox_dpi.setCurrentIndex(index)
else:
    ''' Pref not configured in first use '''
    self.comboBox_dpi.setCurrentIndex(0)
    self.preferences['dpi'] = str(self.comboBox_dpi.currentText())

```

#### **10.3.4 Herència de PyQt**

Les classes de PyQt es divideixen en diversos mòduls, els quals ens interessa:

- QtCore
- QtGui
- QtNetwork
- QtWidgets

Per tal d'extender bé els components o models que s'utilitzen, és necessari heredar-los de la classe principal, concretament una de les anteriors. Ex: from PyQt5.QtWidgets import QProgressDialog

## 10.4 Client

El client de la vista s'utilitza per connectar i parlar directament amb el *core* de l'aplicació, per tal de servir el que la vista demani. És adaptable i escalable, ja que per cada mètode es pot crear una petició al nucli.

La comunicació flueix obrint un socket per comunicar-se amb el connector descrit en ocasions anteriors, i utilitzar l'API per demanar el recurs necessari.

És una idea prou plausible utilitzar un client de vista, ja que imaginem que canviem l'api, s'ha de rescriure la vista, que és molt més molest que reescriure el mètode del client. O com s'ha comentat es pot canviar i utilitzar una altre vista que no sigui la d'aquest projecte, o utilitzar l'aplicació sense interfície gràfica. O fins i tot tenir el nucli de l'aplicació a un servidor i la vista corrent en un altre ordinador. Per definir que és el client, es pot dir que és com una API de la GUI.

Com que el client utilitza l'API, no cal mostrar tots els mètodes, ja que ja en veurem l'aplicació a l'hora de descriure l'API. A continuació es mostra com s'inicia el client i es dóna un exemple de crida a l'api.

```
import zmq
from scannerapp.agent.scann_agent import ScannAgent

class Client(ScannAgent):

    def __init__(self):
        self.subscriber()

    def subscriber(self):
        context = zmq.Context()

        self.sock_subscriber = context.socket(zmq.SUB)
        self.sock_subscriber.connect('tcp://127.0.0.1:5777')
        self.sock_subscriber.setsockopt_string(zmq.SUBSCRIBE, "") #all_
        ↴message from publisher

        self.socket_req = context.socket(zmq.REQ) # socket requester
        self.socket_req.connect('tcp://127.0.0.1:5778')

    def set_preferences(self, preferences):
        if 'path' in preferences.keys():
            if preferences['path'] != None:
                preferences['path'] = preferences['path'].replace('/', '#')
            self.socket_req.send_string('{0}{1}{2}'.format("POST /scanner/", ↴
                preferences, "/preferences"))
            preferences = self.socket_req.recv_json()
```

Notar que en aquest últim mètode canvia el format del path en un de compatible amb l'API. C:/example/path es converteix a C:#example#path. Aquest fet encara fa més visible l'avantatge d'utilitzar un client de vista.

## 10.5 API

L'API definida a l'apartat de disseny s'ha creat utilitzant un llibreria anomenada werkzeug, concretament fent servir els paquets Map i Rule.

Es rep una petició, i s'assigna a una funció passant el mètode i si és el cas els paràmetres, gràcies a les opcions que brinda werkzeug. Es “mapegen” les rutes a un url\_map, i el mètode urls.match “parseja” la ruta i retorna els paràmetres esperats.

Aquest mòdul s'ha anomenat Dispatcher

```
from werkzeug.routing import Map, Rule
from scannerapp.agent.scann_agent import ScannAgent

class Dispatcher(object):
    url_map = Map([
        Rule('/scanners', endpoint='scanners', methods=['GET']),
        Rule('/scanners/<string:scanner_id>/preview', endpoint='preview'),
        Rule('/scanner/<string:preferences>/preferences', endpoint='preferences'),
        Rule('/scan', endpoint='scan'),
        Rule('/scannative', endpoint='scan_native'),
        Rule('/path', endpoint='path', methods=['GET']),
        Rule('/preferences', endpoint='get_pref', methods=['GET', 'DELETE']),
        Rule('/format', endpoint='format', methods=['GET']),
        Rule('/windowID/<int:id>', endpoint='window_id', methods=['POST']),
        Rule('/scanner', endpoint='scanner', methods=['POST'])
    ])

    def __init__(self):
        self.scanner_agent = ScannAgent()
        self.urls = self.url_map.bind('')

    def dispatch(self, url):
        '''dispatch to method request. url.split(' ',1)
        truncate for first space'''
        method, url = url.split(' ', 1)
        endpoint, params = self.urls.match(url, method)
        endpoint = getattr(self, endpoint)
        return endpoint(method, **params)

    def scanners(self, method):
        if method == 'GET':
            _scanners = self.scanner_agent.listscanners()
            return _scanners
        else:
            return []

    def preview(self, method, scanner_id):
        if method == 'POST':
            print("previewing from {}".format(scanner_id))
        else:
            return []
```

```
def scanner(self, method):
    if method == 'POST':
        _scanner = self.scanner_agent.open_scanner()
        return _scanner
    else:
        return []

def preferences(self, method, preferences):
    if method == 'POST':
        import ast
        preferences = ast.literal_eval(preferences)
        for row in preferences:
            _method = row
            value = preferences[row]
            if value != None:
                if _method == 'scann_default':
                    self.scanner_agent.insertScan(value)
                elif _method == 'dpi':
                    self.scanner_agent.insertDpi(value)
                elif _method == 'color':
                    self.scanner_agent.insertColor(value)
                elif _method == 'format':
                    self.scanner_agent.insertOutput(value)
                elif _method == 'path':
                    value = value.replace('#', '/')
                    self.scanner_agent.insertPath(value)
                elif _method == 'delete_files':
                    self.scanner_agent.insertDelete(value)
        return 'OK'
    else:
        return 'ERROR'

def get_pref(self, method):
    if method == 'GET':
        _p = self.scanner_agent.getPreferences()
    elif method == 'DELETE':
        _p = None
        self.scanner_agent.deletePreferences()
    return _p

def scan(self, method):
    if method == 'GET':
        _scan = self.scanner_agent.scan()
        return _scan
    else:
        return []

def scan_native(self, method):
    if method == 'GET':
        _scan = self.scanner_agent.scanNative()
        return _scan
    else:
```

```

        return []

def path(self, method):
    if method == 'GET':
        path = self.scanner_agent.getPath()
        return path
    else:
        return 'ERROR'

def format(self, method):
    if method == 'GET':
        format = self.scanner_agent.getFormat()
        return format
    else:
        return 'ERROR'

def preferences_s(self, method, path):
    path = path.replace("#", "/")
    if method == 'POST':
        self.scanner_agent.insertPath(path)
    return ''

def colour(self, method, colour):
    if colour == 'b_n':
        colour = colour.replace("_", "/")
    if method == 'POST':
        self.scanner_agent.insertColour(colour)
        return 'OK'
    if method == 'DELETE':
        self.scanner_agent.deleteColor(colour)

def scnDefault(self, method, scanner_id):
    scanner_id = scanner_id.replace("_", " ")
    if method == 'POST':
        _scan = self.scanner_agent.insertScan(scanner_id)
        return _scan

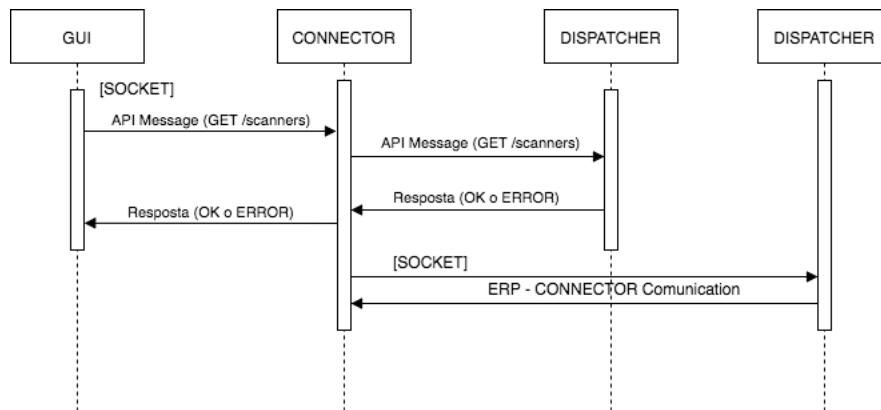
def delete_preferences(self, method):
    if method == 'DELETE':
        _del = self.scanner_agent.deletePreferences()
        return 'OK'
    else:
        return 'ERROR'

def window_id(self, method, id):
    if method == 'POST':
        id = self.scanner_agent.set_window_ID(id)
        return 'OK'
    else:
        return 'VIEW ID ERROR'

```

## 10.6 Connector

El connector té la funció de comunicar i connectar l'ERP amb l'aplicació. A més de rebre i servir les peticions del client. Per fer-ho rebota els missatges que rep del client de la vista i en serveix la petició retornada de l'API. Existeix una comunicació socket a banda i banda. A més a més ha d'obrir la vista quan se li demani. Concretament per detallar-ho gràficament, ho fa de la següent manera:



Per obrir els sockets utilitza ZMQ, de la forma que es defineix en l'apartat corresponent d'aquest treball. Interactua amb els sockets de la següent forma:

```

def config(self):
    parser = configparser.ConfigParser()
    parser.read('config.cfg')
    port = parser.getint('PORT', 'port')
    ip = parser.items('HOST')[0][1]
    protocol = parser.items('PROTOCOL')[0][1]

    return '{0}://{1}:{2}'.format(protocol, ip, port)

def ROUTER(self):
    port = 5772
    addr = '127.0.0.1'
    context = zmq.Context()
    socket = context.socket(zmq.STREAM)
    socket.bind('{0}{1}:{2}'.format('tcp://', addr, port)) # ZMQ does not
    ↵support localhost
    recv = False

    self.logger.info('##### Started')
    while recv != True:
        client_id, message = socket.recv_multipart()
        message = message.decode('utf8')
        if str(message) == 'open':
            self.logger.info('{0}{1}{2}{3}'.format('##### Opened on tcp #',
    ↵', port, 'addr: ', addr))
            recv = True

    self.open_subprocess()
    
```

```

    return (client_id, socket)

def REP(self):
    context = zmq.Context()
    rep_socket = context.socket(zmq.REP)
    rep_socket.bind(self.conn)

    return rep_socket

```

Com que el connector no té interfície gràfica, i cal obrir la vista des d'aquest, un cop s'ha tancat la interfície, cal deixar el programa latent de tal manera que es pugui reconnectar quan es faci un altra petició des de l'ERP:

```

def reopen(self):
    self.router_socket.close()
    self.client_socket, self.router_socket = self.ROUTER()

```

Aquest és el mètode principal per gestionar els missatges amb l'ERP i el client de la vista:

```

def recvMsg(self):
    while True:
        ''' Receiving... '''
        recv = self.rep_socket.recv_string()

        if 'scanned' in recv:
            recv = recv.split(' # ')[1]
            self.rep_socket.send_json('')
        if 'attach_file' in recv:
            recv = recv.split(' # ')[1]
            self.router_socket.send_multipart([self.client_socket, recv.
→encode()])
            # Send bytes type
            rsck_rcv = self.router_socket.recv_multipart() # ok or error
→during attaching
            self.rep_socket.send_json('')
        elif 'close' in recv:
            self.router_socket.send_multipart([self.client_socket, b'close
→'])
            self.rep_socket.send_json('')
            self.close_subprocess()
            self.reopen()
        elif 'trash_file' in recv:
            try:
                recv = recv.split(' # ')[1]
                self._attachments.remove(recv)
                self.rep_socket.send_json('')
            except ValueError:
                self.rep_socket.send_json('')
        else:
            try:
                send = self.dispatcher.dispatch(recv)
                ''' Sending... '''
                self.rep_socket.send_json(send)
            except Exception as exc:

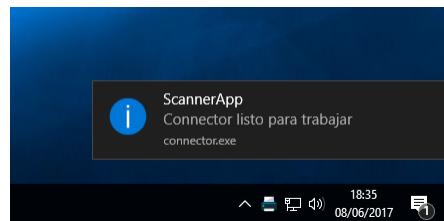
```

```
'''Capture sentry and logger error'''
perror = '{0} {1}'.format("#####SERVER ERROR", str(exc))
self.rep_socket.send_json(perror)
self.sentry_client.captureException()
self.logger.info(perror)
```

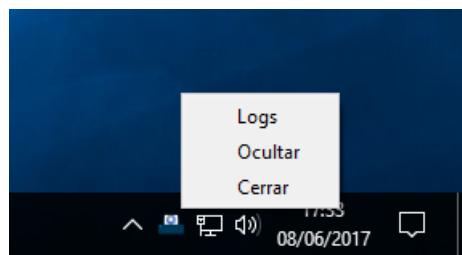
### 10.7 Systray

Quan la classe connector inicia la vista, es té seguretat visible de que el programa està engegat. Però el software incorpora la funcionalitat com s'ha vist anteriorment de quedar-se obert un cop es tanca la vista. Això es fa perquè l'usuari pugui treballar fluidament amb l'ERP, quan sol·liciti l'ScannerApp, s'obre la vista i quan es tanca es realitza el procés automàtic de classificació, però cal tenir el programa latent per si es torna a sol·licitar l'ScannerApp per digitalitzar més documents o realitzar el que calgui. El connector com que no té interfície gràfica, cal proporcionar a l'usuari un feedback de quan està corrent el software o quan està parat. Per fer-ho es pinta una icona a la barra de systray del sistema, i es mostra un missatge quan s'enrega l'aplicació:

Missatge d'inici:



SystrayIcon amb funcionalitats:



Per fer això s'ha creat la següent classe, que hereda un objecte SystrayIcon i permet mostrar i agregar funcionalitats a la icona de programa.

```
import sys
from PyQt5 import QtGui
from PyQt5.QtWidgets import QSystemTrayIcon, QApplication, QMenu, QAction, QMainWindow
```

```

class SysTray(QSystemTrayIcon):
    def __init__(self):
        QSystemTrayIcon.__init__(self)
        self.systray = QSystemTrayIcon(QtGui.QIcon("printer.png"))

    def show(self):
        self.systray.show()
        self.systray.showMessage("ScannerApp", "Connector listo para trabajar",
                                QSystemTrayIcon.Information)

        # Creació menú
        self.systray_menu = QMenu()
        # Afegir options
        self.hide_systray = self.systray_menu.addAction("Ocultar")
        self.close_systray = self.systray_menu.addAction("Cerrar")
        # Connectar senyals amb slots
        self.hide_systray.triggered.connect(self.systray.hide)
        self.close_systray.triggered.connect(self.close)
        # Set context menú
        self.systray.setContextMenu(self.systray_menu)

    def close(self):
        sys.exit()

```

## 10.8 Threading

En crear el SystrayIcon anterior, sorgia un problema: El connector és un procés independent, pel que en iniciar-lo, calia crear un objecte del tipus systrayicon, i seguir amb l'execució. Òbviament ha de córrer com un procés síncron, però el systrayicon està preparat per iniciar-se i quedar-se esperant events fins a finalitzar, tal com passa en les vistes de PyQt. Si s'implementa una classe a part, i s'instància des del connector, aquest systray es queda en espera i no es recupera mai el control del connector, ja que el fil d'execució és en aquest systray. Òbviament es pot integrar tot dins la mateixa classe i definir com a procés principal aquest systrayIcon, però val la pena dedicar un temps a les bones pràctiques. Per tenir els dos processos corrent alhora, què millor que els *threads*. Per tant s'ha creat una petita classe per executar els dos processos en paral·lel i que no es quedí bloquejat un o altre. D'aquesta manera el systray pot passar informació a la classe Pare i mantenir una comunicació síncrona.

```

class Thread(threading.Thread):
    def __init__(self, thread_id, name):
        threading.Thread.__init__(self)
        self.thread_id = thread_id
        self.name = name

    def run(self):
        app = QApplication(sys.argv)
        systray = SysTray()
        systray.show()
        sys.exit(app.exec_())

```

## 10.9 ScannerAgent

L'agent d'escàner s'ha d'encarregar de fer transparent el protocol d'accés, tant si s'utilitza twain com si s'utilitza sane, com qualsevol altre que es pugui afegir o canviar en un futur. És també l'encarregat de gestionar la base de dades del client.

Rep les peticions directament de l'API, és a dir, aquesta última interactua amb l'agent d'escàners per demanar els recursos i les funcionalitats necessàries de forma totalment transparent al protocol que hi ha a l'altre extrem. Es pot fer d'una forma trivial:

```
import sys

if sys.platform == "win32":
    ScannAgent(protocol='Twain')

elif sys.platform.startswith("linux"):
    ScannAgent(protocol='Sane')
```

Al construir l'agent d'escàner, incorpora herència de protocol.

```
class ScannAgent(TwainBase, SaneBase, protocol=None):
    def __init__(self):
        super(ScannAgent, self).__init__()
        self.productName = "SCANNER APP"

        # Initialise TWAIN Base class
        self.path = None
        self.set_default_protocol(protocol)
        self.preferences = dict.fromkeys(['scann_default', 'dpi', 'color',
                                         'path', 'format', 'delete_files'])
        self._methods = ['scann_default', 'dpi', 'color', 'path', 'format',
                        'delete_files']
        self.logger = logging.getLogger('connector')
        self.start_conn()
```

Durant el desenvolupament del projecte s'intenta aprofitar en tot moment avantatges del llenguatge, en aquest cas Python. En el tall de codi següent s'aprofita la funció zip que incorpora el llenguatge que combina els elements de dues estructures de dades formant una llista de tuples de la següent forma:

```
a = [x1, x2, x3]
b = [y1, y2, y3]
>>> zip(a, b)
[
    (x1, y1),
    (x2, y2),
    (x3, y3)
]
```

Això s'aprofita per crear una llista de tuples id\_pref, value\_pref per actualitzar les preferències del protocol. També s'utilitza el mètode setattr comentat anteriorment, però actualitza un atribut o mètode de classe, en comptes d'obtenir-lo.

Mètode per llegir preferències:

```
def loading_preferences(self):
    if self.cursor.execute(self.query_preferences).fetchall():
        params = self.cursor.execute(self.query_preferences).fetchone()
        for x in zip(self._methods, params): # (method, value)
            self._preferences[x[0]] = x[1]
            setattr(self, x[0], x[1])
```

A l'hora d'interactuar amb la base de dades, s'ha utilitzat SQLite3, i ja s'han exposat les característiques anteriorment. Veure apartat: [SQLITE](#)

L'agent d'escàner haurà de gestionar els inputs de totes les dades, a continuació es mostra com s'ha fet, mostrant com a model el canvi de path per guardar els documents:

```
def insertPath(self, path):
    self.protocol.path = path
    self._preferences['path'] = path

    if self.tableIsEmpty():
        self.insertOne('path', path)
    else:
        self.updateOne('path', path)
```

L'agent d'integració d'escaneig parlarà amb el protocol per obtenir documents, llistar escàners... Les funcionalitats que s'han descrit anteriorment, pel que interconnexió amb els escàners es refereix:

```
def preview(self, wid, method):
    self.protocol.get_scanners()

def scan(self):
    pil = self.protocol.scan_tw()

    return pil

def scanNative(self):
    pil = self.protocol.scan_native()

    return pil

def list_scanners(self):
    _scanners = self.protocol.get_scanners()

    return _scanners
```

## 10.10 Protocol Twain

Per implementar el protocol d'accés als escàners, s'ha d'incloure la llibreria twain per Python, (Pytwain) Veure apartat: [Accés a Data Source](#)

Aquest apartat pretén ser una ampliació del capítol de twain, ja que se'n detallen diversos mètodes i algorismes utilitzats en el projecte.

Tal com es mostra, s'instal·la via pip i s'utilitza fent l'import corresponent.

```
pip install pytwain
import twain
```

Com es defineix en el corresponent apartat, per utilitzar twain, és necessari tenir construïda una aplicació bàsica, crear un Source Manager i posteriorment un Source Data.

El Source Manager actua d'intermediari entre l'aplicació i el Source Data (escàner). Són diverses les característiques d'aquest recurs, que principalment permeten obtenir accés a aquests dispositius i interactuar amb ells. Per tant el Source Manager ve a ser un connector principal, el qual a més a més afegeix funcionalitats per obtenir característiques de l'entorn i el sistema. El mètode principal de construcció d'un source manager es fa de la següent manera:

```
self.sm = twain.SourceManager(parent_window=self.window_id,
                               Language=twain.TWLG_SPANISH,
                               Country=twain.TWCY_SPAIN,
                               ProductName='ScannerApp')
```

El constructor del SourceManager accepta molts més paràmetres, però aquí es centren els bàsics i necessaris. L'únic paràmetre obligat és el parent\_window, que serveix per marcar al Source Manager, a quin nivell de finestres ha de llançar els prompts. Empíricament a l'utilitzar twain amb diverses arquitectures, canvia la forma d'instanciar-lo. Ha estat un verdader maldecap debugar i solucionar aquestes restriccions entre arquitectures, fent que hagi calgut "tetejar" i canviar entre controladors d'escàners, Sistemes Operatius, capabilities... No ha estat fàcil obtenir-ne una solució per construir un protocol estable.

A continuació s'exposen els principals algorismes per interactuar amb els escàners, realitzats en aquest projecte.

### 10.10.1 Obrir Source Manager

```
def init_SourceManager(self, MainWindow=None, ProductName=None):
    '''Initialize source manager object'''
    if ProductName:
        self.ProductName = ProductName
    if MainWindow:
        self.MainWindow = MainWindow
    try:
        self.SourceManager = twain.SourceManager(parent_window=self.
                                                MainWindow,
                                                Language=twain.TWLG_
                                                SPANISH,
                                                Country=twain.TWCY_SPAIN,
                                                ProductName=self.
                                                ProductName)
    except twain.excSMOpenFailed as error:
        self.sm = twain.SourceManager(1)
```

### **10.10.2 Llistar escàners**

Twain a través d'un Source Manager retorna una llista d'escàners *Disponibles* al sistema. S'entén com a Disponible:

- Escàner instal·lat i disponible actualment
- Compatible amb twain
- Controladors instal·lats correctament amb l'arquitectura corresponent

Més endavant caldrà tenir compte utilitzar el name\_id d'aquests escàners, ja que twain retorna una llista d'strings, i el mètode per obrir un Data Source espera el name\_id com a tipus bytes, per tant caldrà realitzar la conversió de tipus.

Aquest és el mètode utilitzat al protocol per llistar els escàners:

```
def get_scanners(self):
    '''Return available scanners'''
    if not self.sm:
        self.init_SourceManager()
    try:
        _scanners = self.sm.GetSourceList()
        if _scanners:
            try:
                return _scanners
            except IndexError:
                return ['No scanners data Source']
        else:
            return ['No scanners data Source']
    except twain.excTWCC_NODS:
        return ['No scanners data Source']
```

### **10.10.3 Obrir un Data Source**

Com a post-condició del protocol, a l'obrir un Data Source s'estableix per defecte a l'scannerApp. També es poden obrir diversos Data Source alhora, o utilitzar-los d'altres formes, com es mostrerà més endavant.

```
def get_scanner(self, scan_name):
    '''Return scanner by name'''
    if not self.sm:
        self.init_SourceManager()
    if isinstance(scan_name, str):
        self.sm.open_source(str.encode(scan_name))
    else:
        self.sm.open_source(scan_name)
```

S'ha tingut cura a l'hora de obrir un Data Source fent *casting* a la variable scan\_name i convertir-la a format bytes si és el cas: str.encode(foo\_var).

Anotar que scan\_name ha estat obtingut prèviament. Vegem-ne un exemple de com obrir el primer escàner disponible que ens concedeix el Source Manager:

```
import twain
sm = twaim.SourceManager(0)
sd = sm.open_source(sm.GetSourceList()[0].encode())
```

Un data Source també pot utilitzar-se sense haver-lo obert, però amb els requisits del projecte fa que es necessiti obrir-lo, i a més a més, és un mètode amb poc de control i llarg d'utilitzar ja que cada cop s'ha d'estar controlant quin Data Source s'utilitza, tractar els name\_id...

```
import twain
sm = twaim.SourceManager(0)
twain.acquire(sm.GetSourceList()[0].encode())
```

És de gran importància controlar els mètodes necessaris per tancar correctament el Data Source. Com s'anota en el capítol de twain, s'utilitzen estats del protocol per interactuar entre funcionalitats, i és necessari seguir l'ordre correctament. Un dels requeriments d'aquests estats, és que un cop es fa el procés *d'escaneig*, si no s'indica manualment que el Data Source ha quedat lliure, en interactuar de nou amb aquest, el protocol llança una excepció de seqüència, conforme a què no s'està interactuant correctament i amb ordre respectant els estats. exCTWCC\_SEQERROR

Els mètodes implementats per tancar els Source Data i el Source Manager principals són els següents:

```
def close_scanner(self):
    try:
        self.sd.destroy()
        self.sd = None
    except:
        self.sd = None

def close_source_manager(self):
    try:
        self.sd.destroy()
        self.sm.destroy()
        (self.sd, self.sm) = (None, None)
    except:
        (self.sd, self.sm) = (None, None)
```

### 10.10.4 Obtenir Document

Twain proposa diverses metodologies per obtenir documents, i principalment utilitza dos nivells d'obtenció. Natiu i per fitxer. En l'àmbit d'aquest TFG s'han utilitzat dos algorismes per obtenir un document, a mode d'escanejar amb preferències o amb els controladors natius de cada escàner.

#### Natiu

La metodologia per obtenir un document de forma nativa, implica fer un *request* a twain per informar que es voldrà obtenir un document i que el Data Source està obert i preparat. Essent dos paràmetres els que es poden passar per indicar si es vol mostrar una versió modal dels controladors, una versió finestra, o cap.

```
self.sd.RequestAcquire(1, 0)
```

A continuació cal informar a twain que tot està llest i pot començar amb el procés d'obtenció d'un document via escàner.

```
rv = self.sd.XferImageNatively()
if rv:
    (handle, count) = rv
```

Això retorna un *handle* del fitxer i un comptador de pàgines restants. I finalment es converteix el document

```
twain.DIBToBMFile(handle, path)
```

**nota:** El procés natiu **no** permet l'obtenció de documents amb diferents formats, per fer-ho s'han construït diversos *converters* com els següent, ja que tot i escanejar nativament, es dóna l'opció a l'usuari d'escol·lir el format de sortida:

```
def bmp2pdf(self, file):
    ''' Convert a bmp file to PDF file, and delete old bmp file '''
    img = Image.open(file)
    output = file.replace('.bmp', '.pdf')
    try:
        img.save(output, "PDF", resolution=100.0)
    except TypeError:
        return 'Error on convert PDF'

    remove(file)

def bmp2tiff(self, file, dpi):
    ''' Convert a bmp file to TIFF file, and delete old bmp file '''
    img = Image.open(file)
    output = file.replace('.bmp', '.tiff')
    try:
        img.save(output, "TIFF", resolution=100.0, dpi=(dpi, dpi))
    except TypeError:
        return 'Error on convert TIFF'
    remove(file)
```

Aquest és l'algorisme principal utilitzat en el projecte:

```
def scan_native(self):
    try:
        if not self.sm:
            self.sm = twain.SourceManager(0)

        # Open source & Request for image
        if not self.sd:
            if self.source:
                self.sd = self.sm.open_source(product_name=self.source)
            else:
                self.select_scanner()
```

```
    self.sd.RequestAcquire(1, 0)
    rv = self.sd.XferImageNatively()
    if rv:
        (handle, count) = rv

    self.record_attachment_name()
    path = self.absolute_path.replace(self.format, 'bmp')
    twain.DIBToBMFile(handle, path)
    temp = handle

    self.close_scanner()

except twain.excDSTransferCancelled:
    return 'transfer canceled by user'
except Exception as err:
    return 'transfer error'

self.converters(self.absolute_path.replace(self.format, 'bmp'))

return temp
```

També s'utilitza una `property` de Python, que és un mètode decorador per establir un valor canviant en una variable de classe, per obtenir un nom automàtic del document utilitzant la data del sistema:

```
FILE_FORMAT = '%Y%m%d_%H%M%S' # CONSTANT

@property
def absolute_path(self):
    if self.path:
        if self.format:
            return '{0}/{1}.{2}'.format(self.path, self.attachment_name, self.
                                         format)
        else:
            return '{0}/{1}.{2}'.format(self.path, self.attachment_name, 'bmp'
                                         )
    else:
        from os import environ
        if sys.platform == 'win32':
            if self.format:
                return r'{0}/{1}/{2}.{3}'.format(environ['USERPROFILE'],
                                         'Pictures', self.attachment_name, self.format)
            else:
                return r'{0}/{1}/{2}.{3}'.format(environ['USERPROFILE'],
                                         'Pictures', self.attachment_name, 'bmp')
        else:
            return r'{0}/{1}.{2}'.format(environ['PATH'], self.attachment_
                                         name, 'bmp')

    def record_attachment_name(self):
        self.attachment_name = '{0}_{1}'.format('adjunto', datetime.now().
                                         strftime(FILE_FORMAT))
```

## Amb preferències

Escanejar indicant manualment les preferències d'usuari, es pot realitzar de diverses maneres. Però la més interessant i funcional tot i que una mica més laboriosa es fa “obrint capabilities”, tal com es mostra en capítols anteriors. Indicant-ne així un estat virtual del Data Source perquè faci la feina d'una manera o un altre.

S'han creat diccionaris com els següents per indicar els valors de les capabilities:

```
COLORS = {'bw': twain.TWPT_BW,
          'gray': twain.TWPT_GRAY,
          'color': twain.TWPT_RGB}

OUTPUT_FORMATS = {'bmp': twain.TWFF_BMP,
                  'pdf': twain.TWFF_PDFA,
                  'tiff': twain.TWFF_TIFF,
                  ...}
```

Establiment d'algunes capabilities:

```
if self.color:
    _color = COLORS[self.color]
    self.sd.set_capability(twain.ICAP_PIXELTYPE, twain.TWTY_UINT16, _color)
if self.dpi:
    self.sd.set_capability(twain.ICAP_XRESOLUTION, twain.TWTY_FIX32, self.dpi)
    self.sd.set_capability(twain.ICAP_YRESOLUTION, twain.TWTY_FIX32, self.dpi)
```

Un cop establertes les capabilities, s'ha d'informar a twain que el Data Source està llest i es vol escanejar via `request`, `self.sd.RequestAcquire(0, 0)`. A continuació cal passar una tupla (`path, format`) a la funció `file_xfer_params`. Tot seguit es canvia d'estat i s'informa a twain que comenci el procés d'obtenció del document amb el mètode `XferImageByFile()`. A continuació es mostra l'algorisme principal:

```
def scan_tw(self):
    try:
        _path = self.absolute_path.replace(self.format, 'bmp')

        if not self.sm:
            self.init_SourceManager()

        self.sm.SetCallback(self.subscribe_twain)

        if self.scanner:
            self.sd = self.get_scanner(self.scanner)
        else:
            self.sd = self.sm.open_source()

        self.sd.RequestAcquire(0, 0)

        if self.format:
            _format = OUTPUT_FORMATS[self.format]
        else:
            # default format
```

```
_format = twain.TWFF_BMP

params = (_path, _format)
self.sd.file_xfer_params = params
self.set_capabilities()

file = self.sd.XferImageByFile()
twain.CancelAll()
except twain.excDSTransferCancelled:
    # usuari ha cancelat l'escanejada
    self.close_scanner()
    return 'transfer canceled by user'
except twain.excTWCC_BUMMER:
    # es solicita massa ràpid l'escàner
    self.close_scanner()
    return 'transfer canceled'
except Exception as err:
    file = err
    pass

self.close_scanner()

return n
```

## 10.11 Integració amb un ERP

L'ERP de l'empresa ha de permetre interactuar amb l'ScannerApp, rebre els documents i adjuntar-los de forma intel·ligent.

**El mode de treball actual és el següent:**

- Usuari treballa amb l'ERP
- Usuari selecciona el plugin de l'ScannerApp
- Treballa amb l'ScannerApp
- Acabada la feina, si hi ha documents, s'envien a l'ERP, i es classifiquen de forma automàtica

Per fer això crea un nou plugin al mòdul de plugins de l'ERP, que controlarà aquesta interacció, comunicant-se de forma síncrona mitjançant un socket amb l'ScannerApp. S'ha proposat les següents crides en funció de la direcció:

**ERP - ScannerApp:** obrir(): el plugin de l'ERP detecta que es vol utilitzar l'ScannerApp (així ho ha indicat l'usuari a través d'un botó), i demana a l'ScannerApp que s'obri. S'estableix la connexió.

message\_process(): ERP informa a ScannerApp de l'estat del procés (adjuntat i classificat correctament o Error)

**ScannerApp - ERP:** attach(): ScannerApp envia un o diversos documents perquè l'ERP els classifiqui. La comunicació ha de ser síncrona, esperant-se a rebre feedback per part de l'ERP

abans d'enviar un altre document.

**close()**: L'usuari ha acabat de treballar amb l'ScannerApp. L'ScannerApp i l'ERP es tanca la connexió.

**nota:** La comunicació és via sockets TCP, orientats a la connexió.

Definició del plugin a l'ERP: Permet crear un nou plugin i definir-lo en el menú de plugins de l'ERP.  
(Importa el mòdul del nou plugin)

```
import re

import workflow_print
import scanner

plugins_repository = {
    'scan': {'model':'.*', 'string':_('Scan'), 'action': scanner.scan},
}

def execute(datas):
    result = {}
    for p in plugins_repository:
        if not 'model_re' in plugins_repository[p]:
            plugins_repository[p]['model_re'] = re.compile(plugins_
→repository[p]['model'])
        res = plugins_repository[p]['model_re'].search(datas['model'])
        if res:
            result[plugins_repository[p]['string']] = p
    if not len(result):
        common.message(_('No available plugin for this resource !'))
        return False
    sel = common.selection(_('Choose a Plugin'), result, alwaysask=True)
    if sel:
        plugins_repository[sel[1]]['action'](datas)
    return True
```

Implementació del plugin: Es passa un datas rebent la informació de la secció de la classificació, de tal manera que es podrà crear un document adjunt en base64 en el model de dades indicat, del document que ha enviat l'ScannerApp

```
from __future__ import with_statement
import rpc
import base64
import socket
import errno
from time import sleep
import traceback
import common
import configparser

def scan(datas):
    try:
        if datas['id']:
            parser = configparser.ConfigParser()
```

```

parser.read('config.cfg')
port = parser.getint('PORT', 'port')
host = parser.items('HOST')[0][1]
protocol = parser.items('PROTOCOL')[0][1]

buffer = 4096
socket_client = socket.socket(socket.AF_INET, type=socket.SOCK_
↪STREAM, proto=0)
socket_client.connect((host, port))
socket_client.setblocking(0)

socket_client.send('open')
while True:
    try:
        recv = socket_client.recv(buffer)
    except socket.error, e:
        err = e.args[0]
        if err == errno.EAGAIN or err == errno.EWOULDBLOCK:
            sleep(1)
            # No data to recv
            continue
        else:
            traceback.print_exc()
            return
    else:
        if recv == 'close':
            socket_client.close()
            break
        elif datas['id']:
            try:
                content = recv.rsplit('/', 1)[1] # receive_
↪document

                with open(recv, 'rb') as f:
                    lines = f.read()
                res = rpc.session.rpc_exec_auth(
                    '/object', 'execute', 'ir.attachment', 'create'
↪',
                    {
                        'name': content,
                        'res_model': datas['model'],
                        'res_id': datas['id'],
                        'datas': base64.b64encode(lines)
                    }
                )
                socket_client.send('attached ok')
            except Exception:
                socket_client.send('error while attaching')

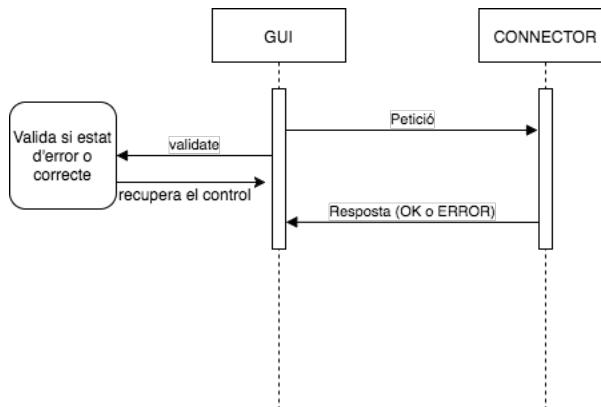
        else:
            common.warning('You must resource a object', 'Warning')
    except Exception:
        traceback.print_exc()

```

## **10.12 Control d'errors**

Els dos scripts de Python a executar (GUI i connector) tindran la decisió sobre el control d'excepcions. Generalment la vista fa peticions al connector, que serà qui retornarà la resposta amb el desitjat o amb un codi d'error.

Veure diagrama:



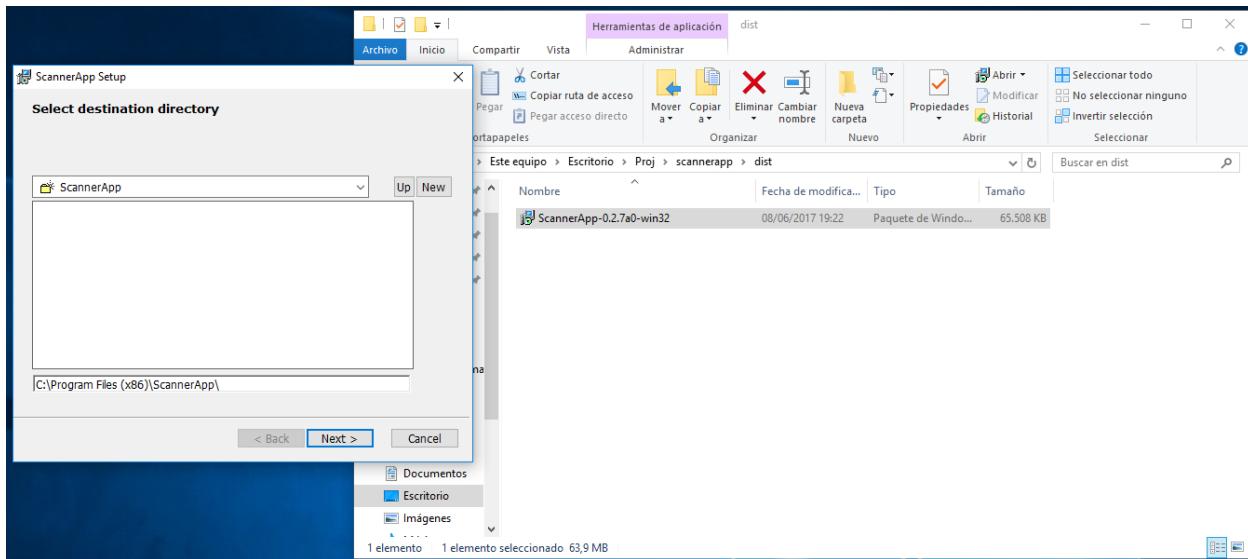
## **10.13 Resultats i proves**

Per realitzar les proves i tests d'aquest projecte, s'ha fet una llista de proves a cada funcionalitat que s'afegia al sistema. Com es comenta en altres apartats, el sistema de tests ha de ser exhaustiu, i una funcionalitat no entrerà a versió fins que no s'hagi provat del tot i certificat que compleix les validacions. Per fer els tests, s'inclou en aquesta llista de proves totes les característiques que ha de complir la funcionalitat. Per ser curosos, a més s'han inspeccionat funcionalitats cap enrere, és a dir, a cada funcionalitat s'han hagut de fer les proves d'altres funcionalitats de més edat, per certificar que aquesta nova no canvia el comportament de l'anterior. És a dir, suposem que una funcionalitat A ha estat provada i agregada correctament. Ara es desenvolupa una funcionalitat B, la qual s'ha acabat el procés de desenvolupament i es passa al procés de test. El procés de test d'aquesta funcionalitat B ha de complir: \* llista de proves de la funcionalitat B \* llista de proves de la funcionalitat A

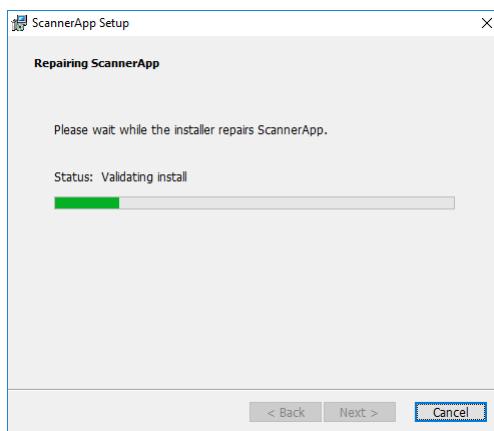
Els requisits de l'aplicació i el temps han fet que no hagi estat possible implementar la metodologia TDD que es proposava en un inici com a objectiu.

Per mostrar els resultats d'una forma més dinàmica, a continuació s'exposarà la funcionalitat, i es mostrerà el resultat

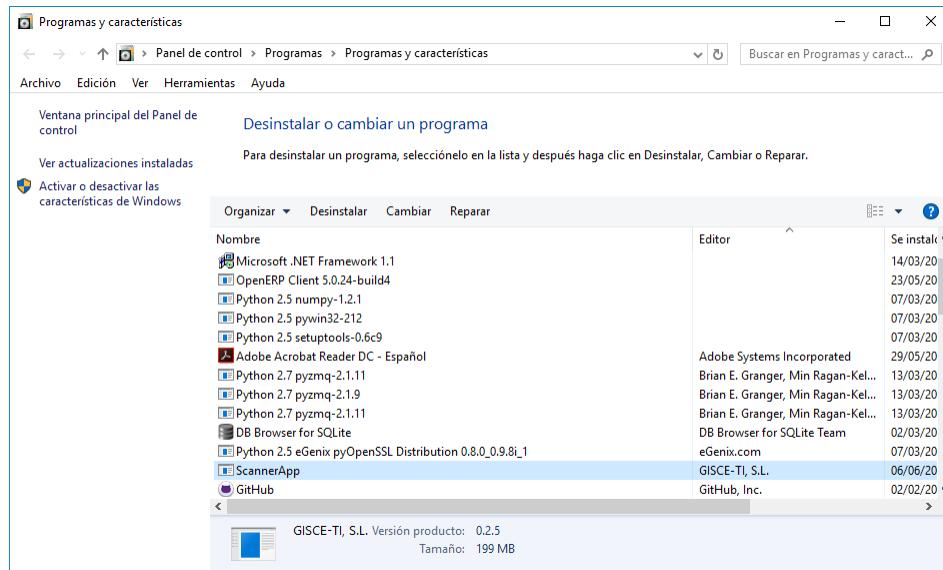
**Instal·lació ScannerApp:** L'instal·lador és un paquet .msi, que es pot instal·lar fent doble clic sobre el mateix. La versió actual és: 'ScannerApp-0.2.7a0-win32.msi'.



Un cop executat s'inicia el procés com es mostra en la següent imatge:

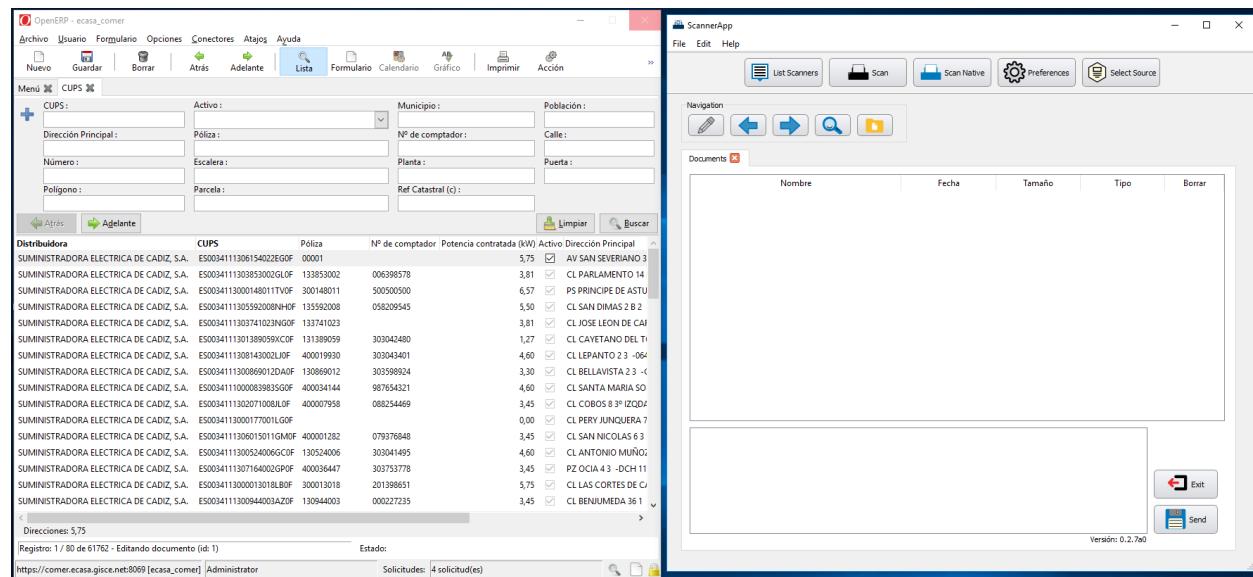


L'instal·lador no demana informació extra, només si es vol escollir un path d'instal·lació, si l'usuari no el canvia, per defecte s'instal·la a C:\Program Files o C:\Program Files (x86), segons el sistema. Finalitzat el procés, el programa queda instal·lat. Per desinstal·lar-lo es pot fer amb l'assistent típic del sistema.



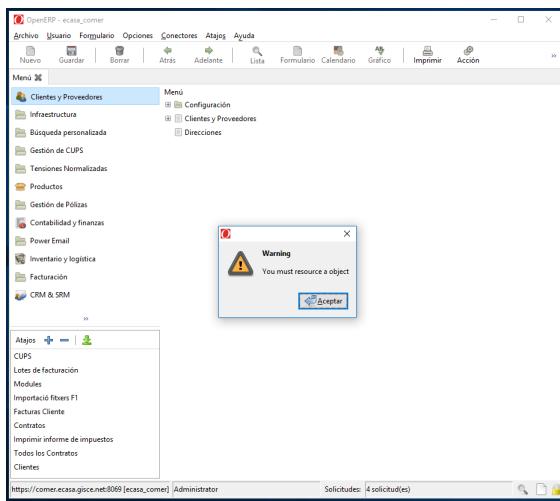
**Treballar amb ScannerApp des de l'ERP:** L'usuari pot iniciar l'ScannerApp seleccionant el botó *Connexores - Scan*. Cal recordar que els documents es guarden directament al recurs actual en el moment d'iniciar l'ScannerApp. És a dir si s'està en una pòlissa, i l'usuari inicia l'ScannerApp, el documents que obtingui amb aquest, quedaran adjuntats i classificats en aquesta pòlissa. Per iniciar l'aplicació, cal estar dins un recurs de l'ERP (La pòlissa d'un client, facturació, CUPS...). En definitiva, recursos típics de les empreses elèctriques. Els modes de selecció suportats són: mode llista i mode formulari. Es pot iniciar l'aplicació si s'està dins un recurs, o des de la llista de recursos, però havent-lo seleccionat amb el ratolí.

Un cop s'ha fet clic al botó *Scan*, s'inicia el programa tal com es mostra en la següent imatge:

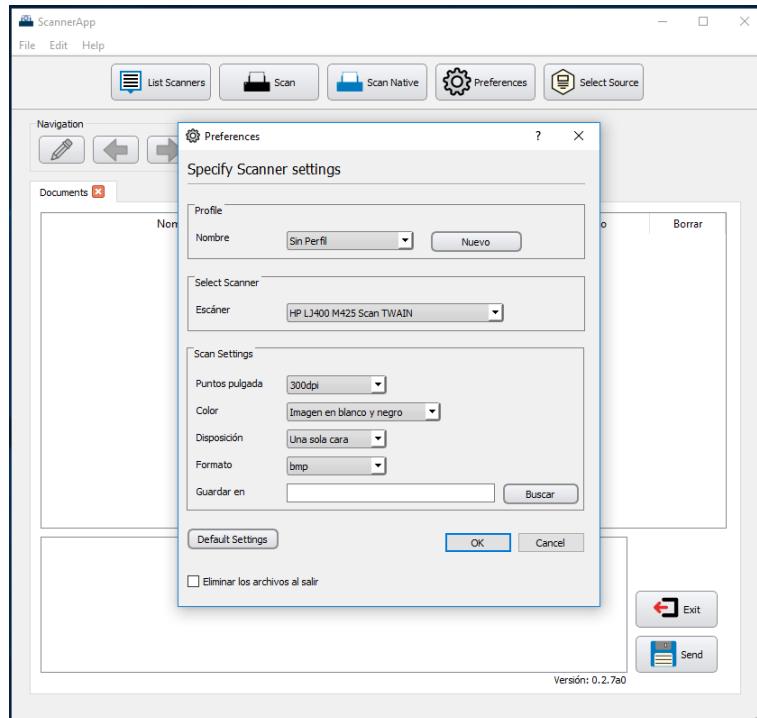


Si l'usuari no es troba dins de cap recurs de l'ERP, l'aplicació llança un missatge d'informació notificant-ho

tal com es pot veure en la imatge que precedeix:



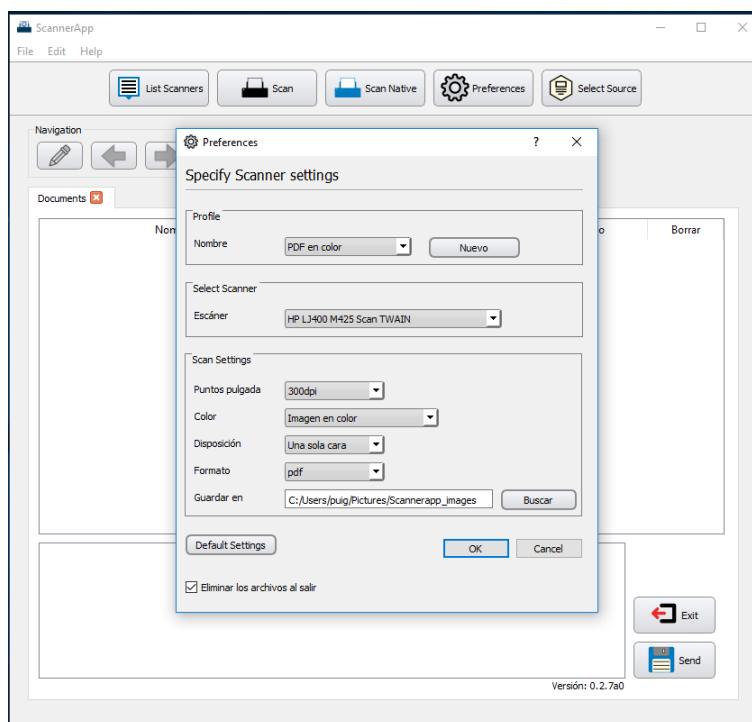
**Preferències i perfils** L'usuari pot definir preferències per digitalitzar documents, que s'utilitzaran en el procés de digitalització (botó *scan*). Es pot seleccionar el botó *Preferences*, o navegar amb el menú superior (*File - preferences*), per obrir una finestra de configuració com la següent:



L'usuari pot gravar les preferències, cancel·lar l'estat actual, reiniciar-les per defecte, o establir perfils de digitalització. L'usuari pot seleccionar:

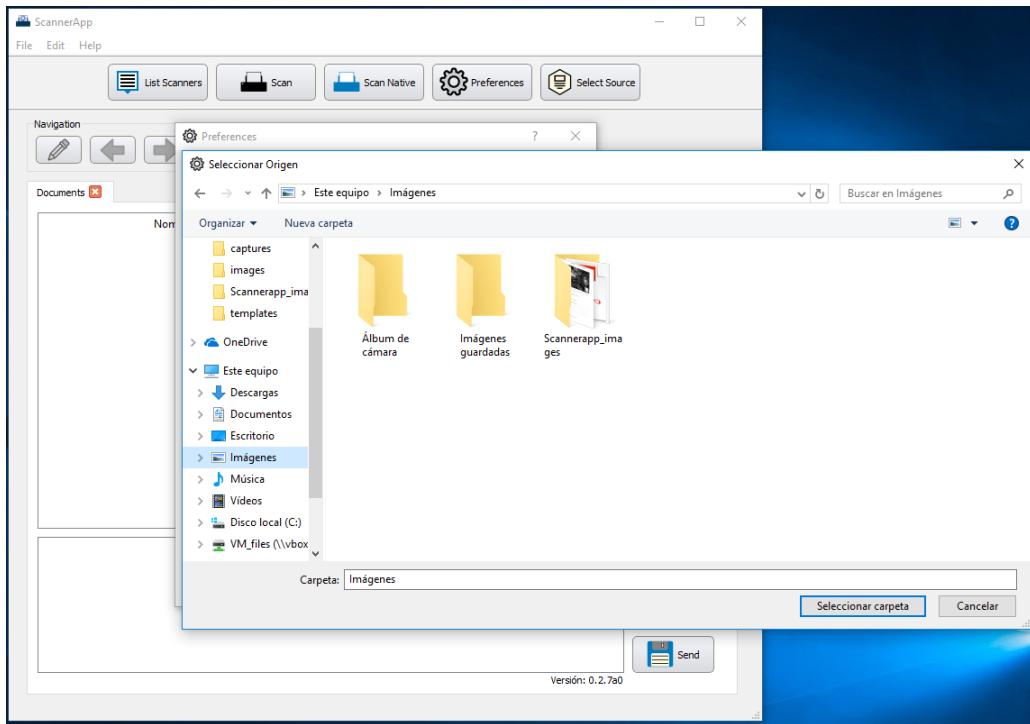
- Perfil
- Escàner: predefinir un escàner per defecte
- Resolució
- Color: color, escala de grisos, blanc i negre
- Una o doble cara
- Format: format de sortida del document. Típics: pdf, bmp i tiff
- Directori: directori de l'ordinador físic on es guarden els documents. Si l'usuari no defineix cap directori per defecte, l'aplicació sel
- Eliminar arxius al sortir: selecciona el mode d'esborrat dels documents, que s'eliminen de l'ordinador un cop s'han classificat a l'ERP.

La següent imatge mostra una configuració típica de preferències de digitalització:



Per seleccionar les preferències es fa mitjançant les llistes desplegables d'aquesta finestra. Per seleccionar el directori on es guardaràn les imatges, es pot fer utilitzant el botó *buscar*, que obra una finestra de navegació típica del sistema, per seleccionar el directori. Per reiniciar les preferències a l'estat per defecte cal fer clic al botó *Default Settings*. Per establir un nou perfil amb les preferències entrades a la pantalla, cal fer clic al botó *Nou* de l'apartat *Profile*, i indicar un nom de referència.

La següent imatge mostra com escollir un directori on guardar els documents:

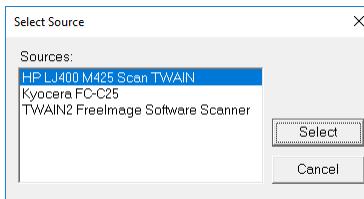


Si l'usuari no defineix cap directori, per defecte l'aplicació guarda els documents al directori Imatges de l'usuari.

Un cop gravades les preferències, queden guardades a la base de dades, per tenir-les disponibles en els següents usos de l'ScannerApp.

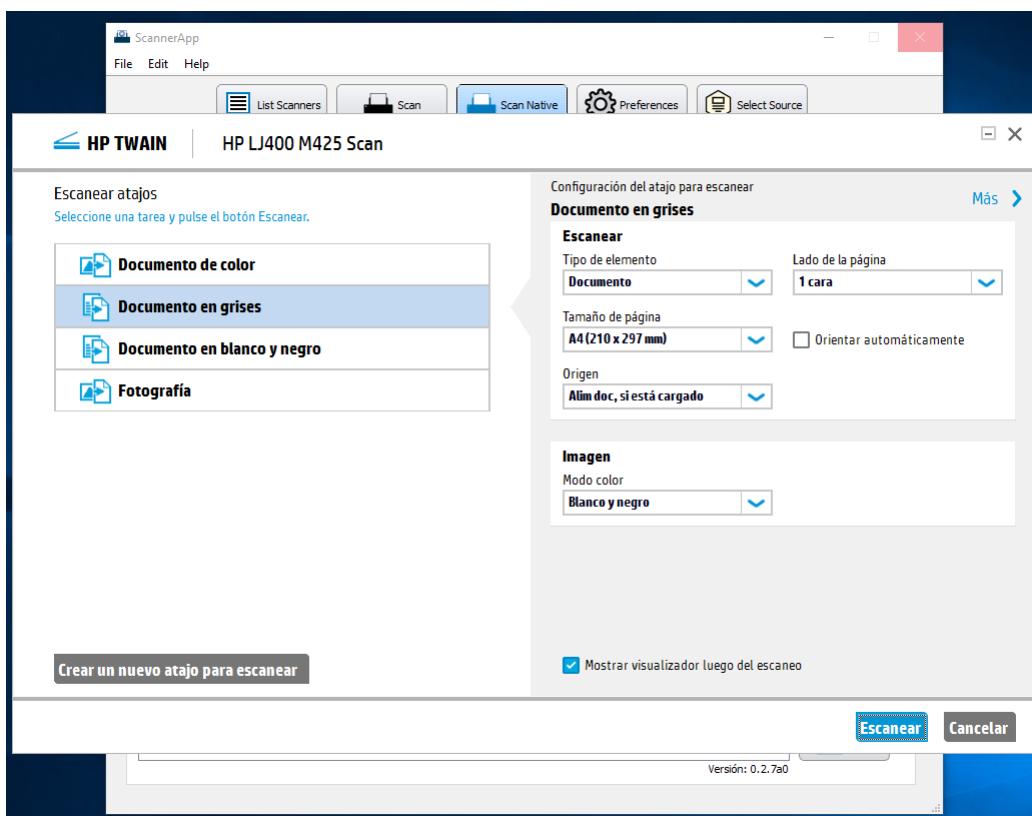
scann_default	dpi	coloring	path	format	delete_files
Filtro	Filtro	Filtro	Filtro	Filtro	Filtro
1 HP LJ400 M42...	300	color	C:/Users/puig...	pdf	1

**Seleccionar escàner per defecte** A més de seleccionar l'escàner per defecte des de preferències, l'usuari pot fer clic al botó *Select Source* de la vista principal, per predefinir que s'utilitzí un escàner per defecte. Al fer clic a aquest botó, apareix una finestra com la següent per seleccionar l'escàner:

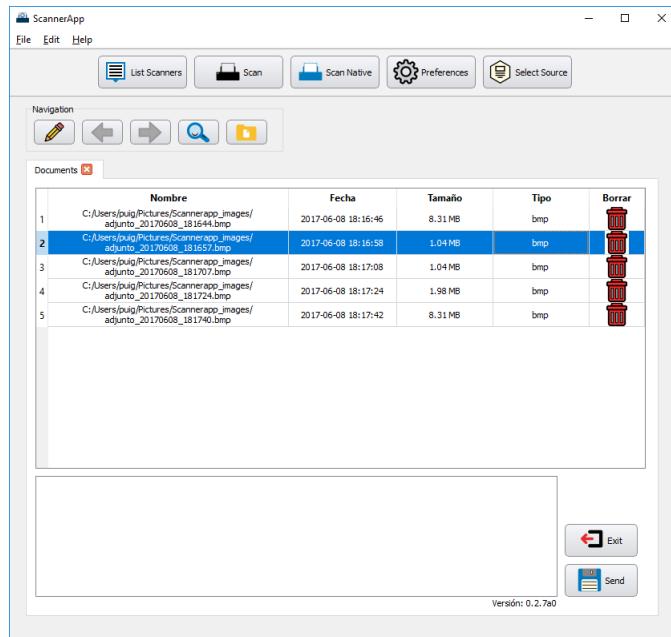


Un cop seleccionat, l'escàner queda establert com a predefinit.

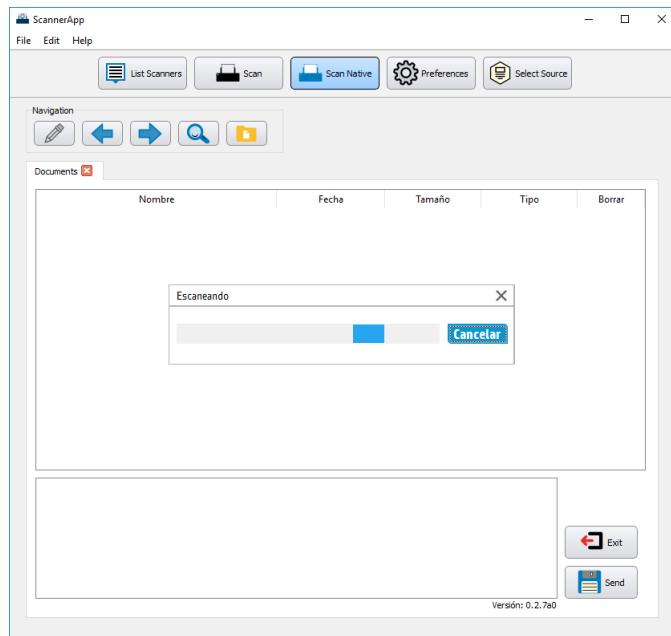
**Procés d'escaneig Natiu** S'inicia el procés seleccionant el botó *Scan Native*. Es presenta a l'usuari els controladors de l'escàner actual tal com es mostra a la imatge que precedeix:



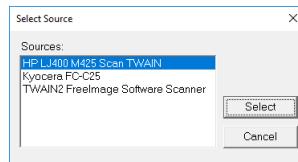
Un cop finalitzat el procés d'escaneig, el document obtingut es guarda al directori definit, i es mostra a la vista principal de la mateixa manera que la imatge següent:



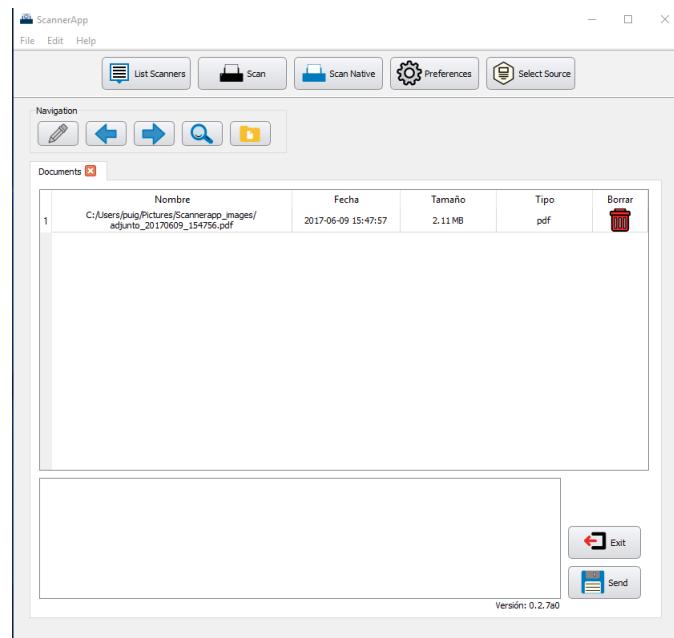
**Procés d'escaneig** S'inicia el procés seleccionant el botó *Scan*. L'aplicació no demanarà res, escanejarà amb les preferències que ha definit l'usuari. Un cop s'ha fet clic al botó *Scan* ja s'inicia el procés com mostra la següent figura:



Si l'usuari **no** ha definit mai les preferències: L'aplicació selecciona les per defecte, i mostra a l'usuari una finestra com la següent per escollir l'escàner (ja que no s'han definit mai les preferències):

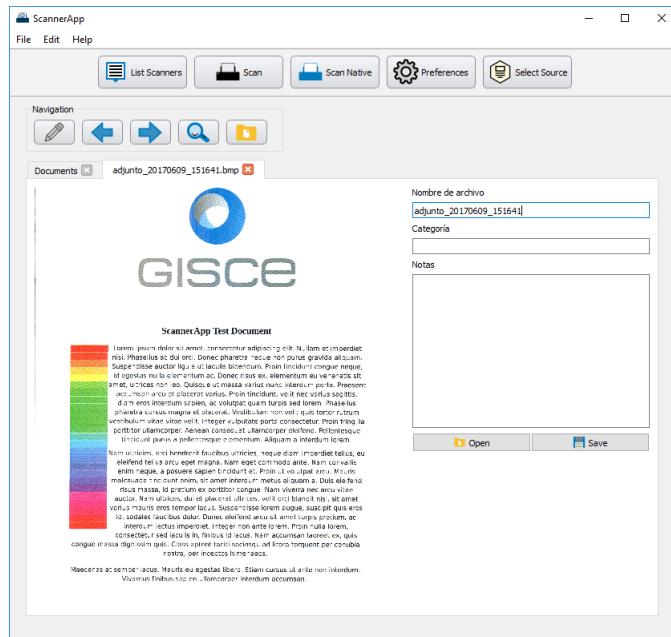


Un cop finalitzat el procés d'escaneig, el document obtingut es guarda al directori definit, i es mostra a la vista principal de la mateixa manera que la imatge següent:

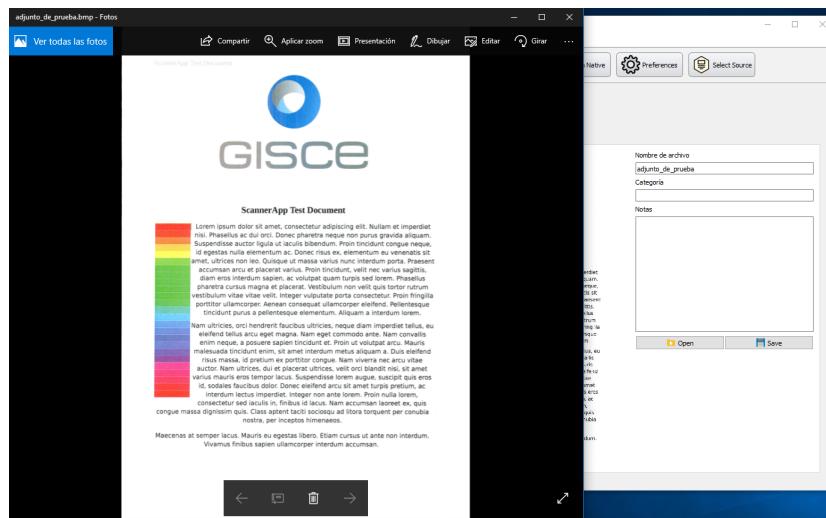


**Navegació i edició de documents** L'usuari pot navegar entre els documents mitjançant un sistema de pestanyes. Per fer-ho, cal fer clic sobre el botó amb la icona del llapis, o simplement fer doble clic al document desitjat de la llista de documents (finestra principal).

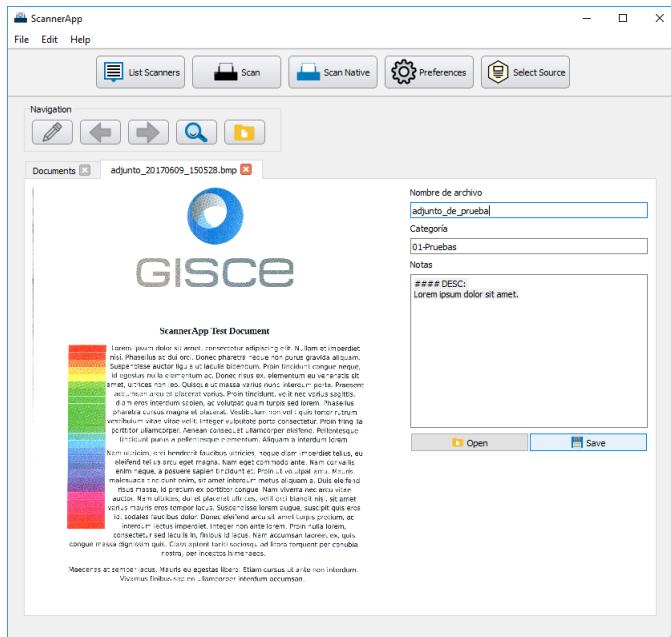
Un cop seleccionat el document, s'obre una nova pestanya com la que mostra la següent imatge:



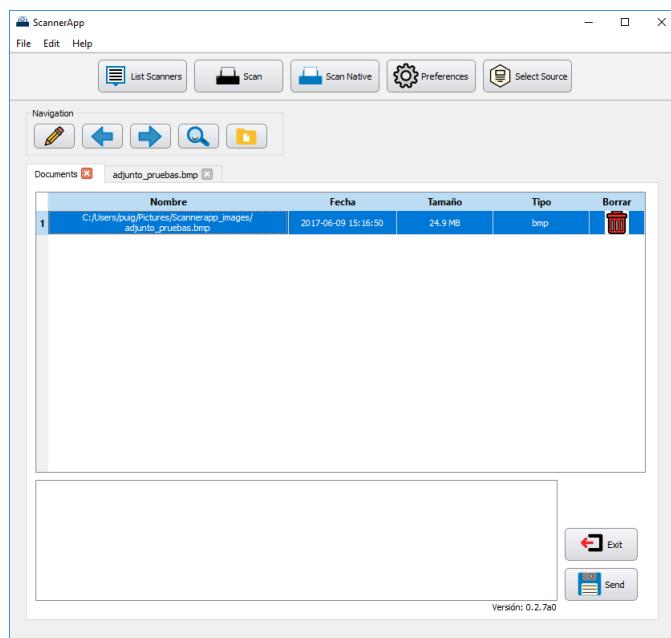
L'usuari pot canviar de pestanya o tancar-la quan vulgui. La funcionalitat de tancar la pestanya principal està desactivada, per tant no es podrà tancar. Un cop obert el document, se'n mostra la previsualització, i si es vol, fent clic al botó *Open*, el document s'obra en format gran, amb l'aplicació predefinida del sistema tal com es mostra a la següent imatge:



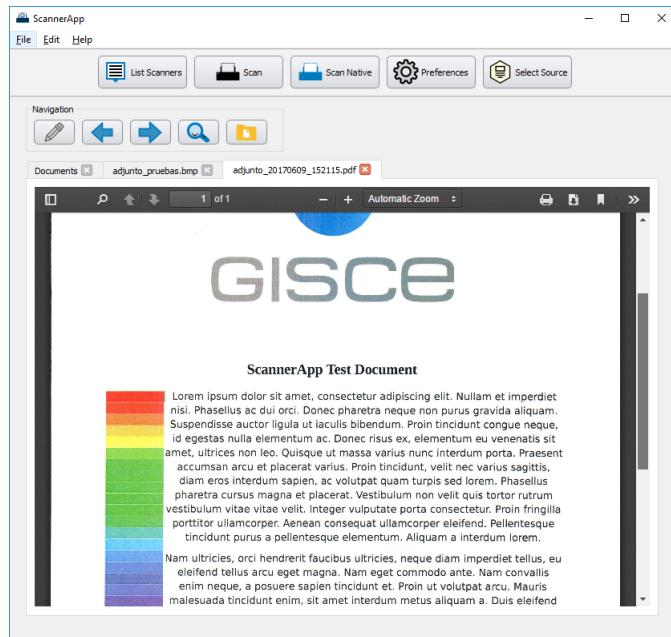
Es pot canviar el nom del document, afegir-hi una categoria, i/o afegir-hi notes. Per guardar els canvis cal fer clic al botó *Guardar*:



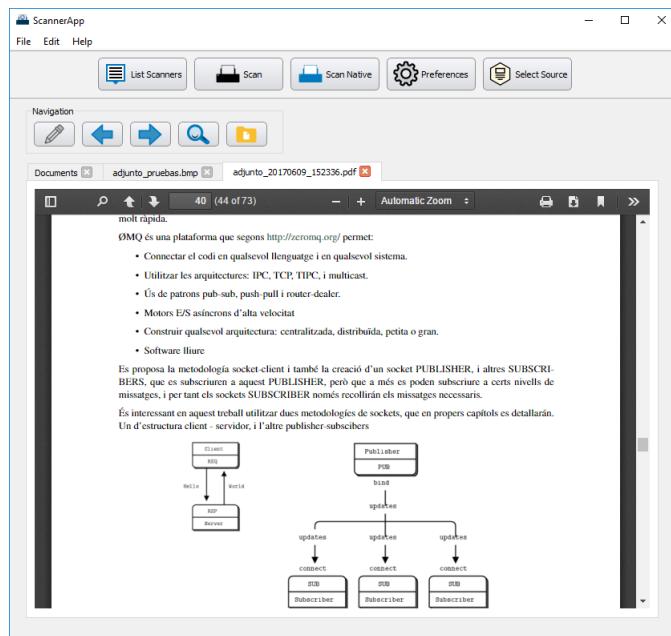
El canvi queda guardat i l'aplicació torna automàticament a la vista principal, tal com es mostra a la imatge:



Si es vol previsualitzar documents d'altres tipus com pdf, es pot seguir el mateix procés amb el botó llapis o doble clic al document. L'aplicació presenta a l'usuari un previsualitzador de documents enriquit com el que es mostra a continuació:



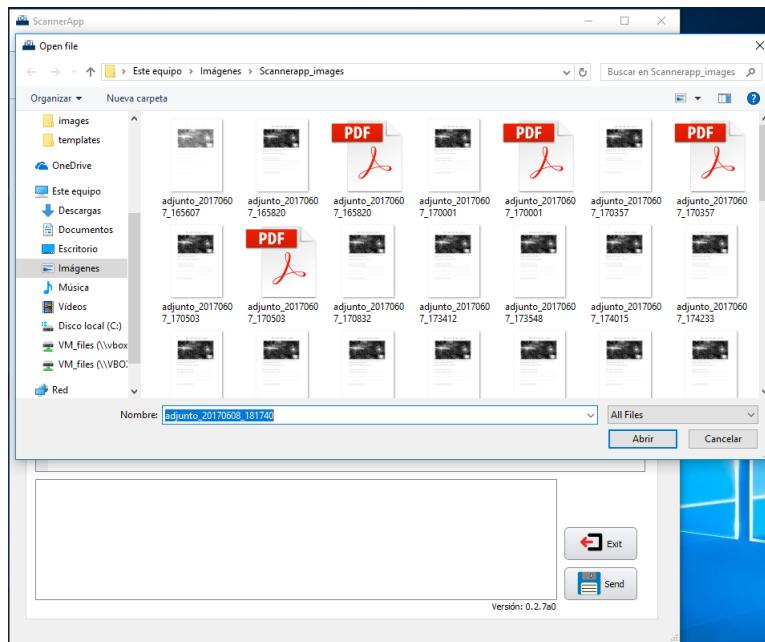
Aquest previsualitzador incorpora funcionalitats de zoom, multipàgina, impresió directe, descàrrega de document, etiquetes, edició...



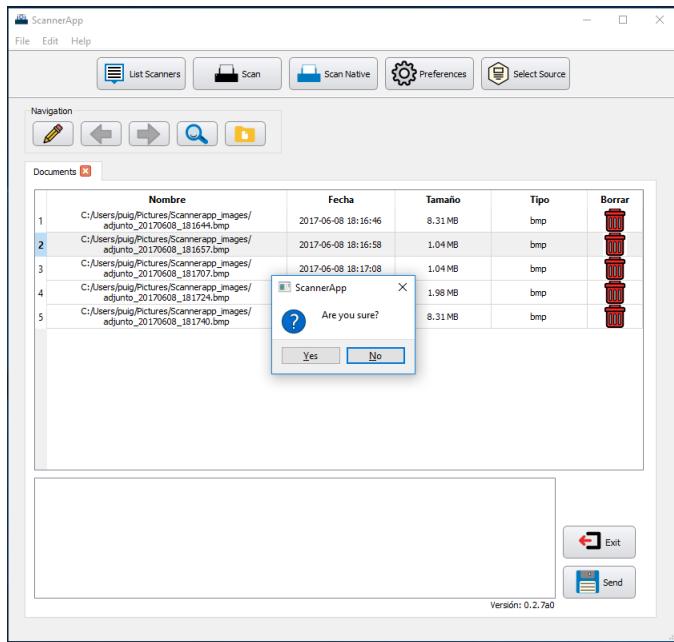
**Navegació** Els botons de navegació són per:

- Llapis: previsualitzar i editar document
- Fletxes: moure's entre documents (és un extra, ja que es pot navegar fent clic a la pestanya corresponent)
- Lupa: tornar a la finestra principal. Els usuaris de l'ERP estan acostumats a aquesta icona per retornar al format llista.
- directori o carpeta: obrir documents

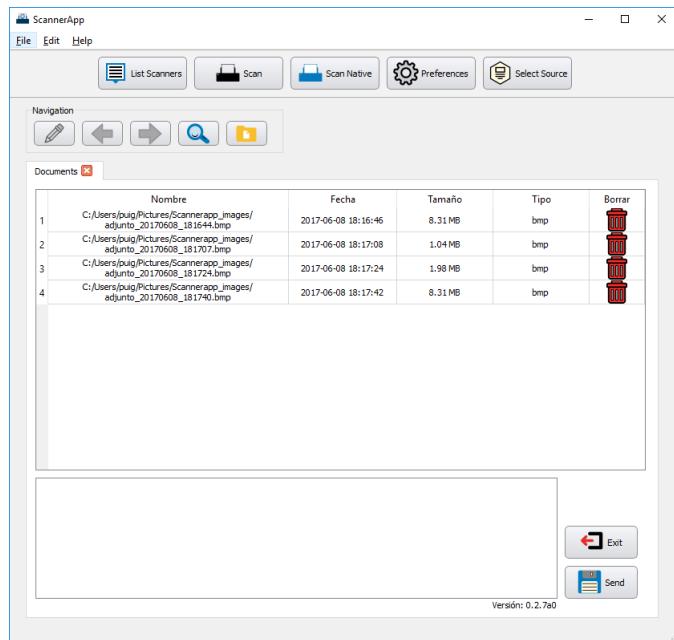
A continuació es mostra com obrir un document en format gran un cop s'ha fet clic al botó amb la icona directori o carpeta:



**Eliminar documents** Si es vol eliminar un document que ha sortit malament, es pot fer mitjançant el botó amb la icona de la paperera, que hi ha al costat de cada document a la vista principal, tal com es mostra a la imatge:

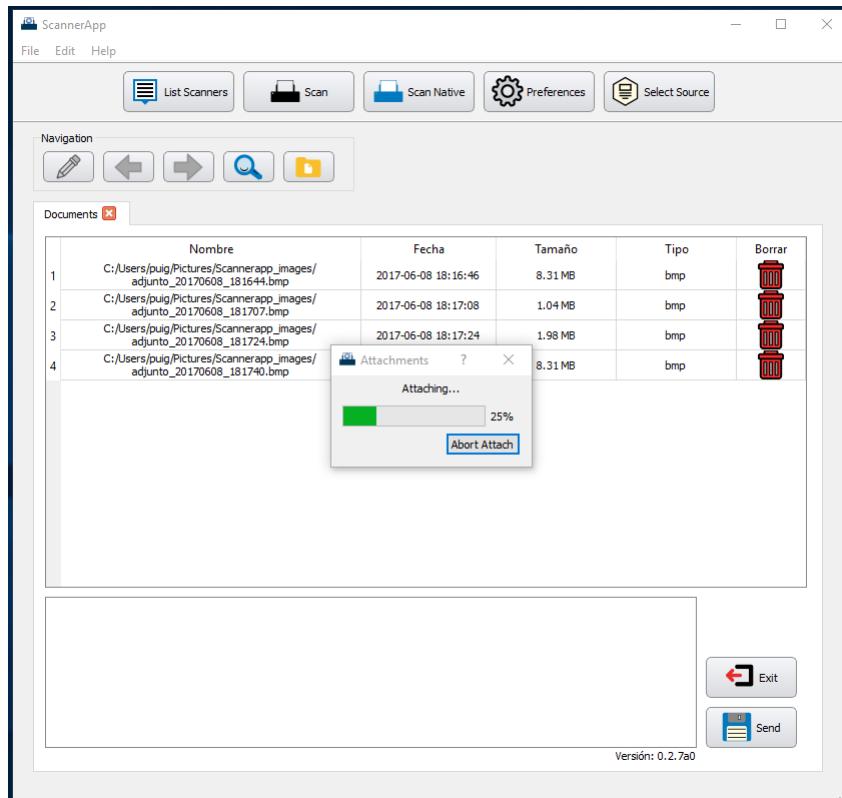


Si l'usuari ha confirmat que vol eliminar el document, aquest s'esborra tal com mostra la següent imatge:



**Finalitzar la feina** Un cop s'ha acabat amb l'obtenció dels documents, i es vol tornar a l'ERP per seguir treballant o obtenir més documents per un altre recurs, l'usuari pot fer clic al botó *Save*, o simplement tancant la finestra del programa. Si no es vol guardar la feina, es pot utilitzar el botó *Exit*. Encara que es tingui activada la preferència d'eliminar arxius al sortir, l'ús d'aquest botó guarda una còpia dels documents per si decás. Les demés funcionalitats no guarden els arxius si l'usuari ho ha predefinit així.

Quan es fa clic al botó *Save* o es tanca el programa, els documents s'adjunten i es classifiquen automàticament dins del recurs amb el qual s'ha iniciat l'ScannerApp. La següent imatge mostra el procés de pujada dels documents.

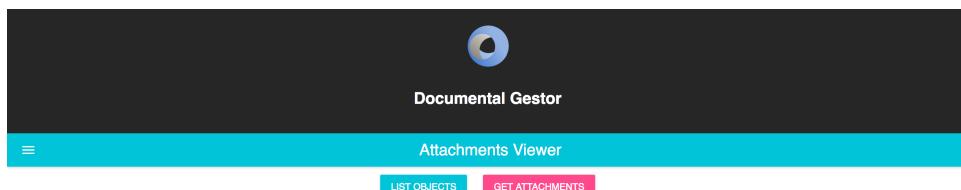


Un cop finalitzat el procés de càrrega, els documents queden adjuntats, classificats, i disponibles des del recurs de l'ERP tal com mostra la següent imatge:

## Software d'integració d'escaneig i gestió documental, Versió Memòria

The screenshot shows a software window titled "OpenERP - ecasa\_comer". The menu bar includes "Archivo", "Usuario", "Formulario", "Opciones", "Conectores", "Atajos", and "Ayuda". Below the menu is a toolbar with icons for "Nuevo", "Guardar", "Borrar", "Atrás", "Adelante", "Lista" (which is selected), "Formulario", "Calendario", "Gráfico", "Imprimir", and "Acción". A sub-menu bar below the toolbar includes "Menú", "CUPS", and "Adjuntos". The main area displays a table with columns: "Nombre del documento adjunto", "Nombre del archivo", "Creador", "Fecha de creación", "Categoría", and "Objeto Asociado". The table lists several attachments, all created by "Administrator" on "08/06/2017" at various times between 09:48:38 and 18:23:07. All attachments are categorized as "OTROS" and are associated with "giscedata.cups.ps: ES0034111306154022EGOF". At the bottom of the screen, there is a status bar with the text "Registro: 9 / 12 de 12 - Editando documento (id: 13727)", "Estado:", the URL "https://comer.ecasa.gisce.net:8069 [ecasa\_comer]", "Administrator", "Solicitudes: 4 solicitud(es)", and some small icons.

**Web** Per ocupació de l'aplicació principal, la web ha quedat fora de l'abast d'aquest treball. S'ha començat amb React.js i s'ha iniciat una interfície bàsica feta amb components de react i material-ui de Google tal com es mostra a la següent imatge:



# CAPÍTOL 11

---

## Treball futur

---

Com es ve comentant en aquesta memòria, aquest projecte està parlat i decidit que continuï, ja des d'un principi. Atès que és un producte que es servirà als clients de l'empresa on s'ha realitzat, és de ben segur que es continua amb l'afegiment de funcionalitats i millores.

En aquest apartat es podria definir una sèrie de millores possibles, o analitzar el que s'hi podria fer a l'aplicació, però de fet l'acord amb l'empresa i el client fa que ja hi hagi molts objectius i tasques que es continuen desenvolupant un cop acabat aquest projecte.

Per donar-les a conèixer a continuació es llisten les tasques que actualment ja estan en procés de desenvolupament o que en un futur pròxim es començaran a implementar:

- OCR amb filtres i processament de dades: analitzar documents i automatitzar processos llegint-los amb un algorisme de reconeixement de fitxers i actuar en conseqüència segons unes directrius marcadess.
- Refrescar pantalla ERP: utilitzar un sistema de refresh automàtic un cop finalitzat el procés amb l'ScannerApp.
- Antimalware de documents
- Implementar perfils i preferències a l'ERP
- Integrar core de l'ScannerApp a servidor i utilitzar la vista en local
- Protocol Sane: implementar el protocol Sane per la comunicació amb els escàners en versions Unix o Mac OS X
- Instal·lar ScannerApp com a servei de Windows
- Web: acabar la web i crear un espai navegable i funcional per la consulta dels adjunts
  - funcionalitats web: incloure les funcionalitats que sorgeixin a necessitat
  - Versió smartphone

- Altres millores

Actualment el TFG ha finalitzat a la versió de software v0.2.7a0.

# CAPÍTOL 12

---

## Conclusions

---

A nivell personal ha estat molt satisfactori desenvolupar aquest TFG. Com es venia explicant a l'inici d'aquest document, m'agrada intentar buscar coses que em motivin, per a mi és molt important, i aquest TFG ho ha aconseguit.

Sempre havia pensat o imaginat que fer un TFG era especial, i havia de fer-lo sobre alguna cosa important, alguna cosa que m'estimulés, i ara veure que el programa ja està desplegat a casa el client, i com l'utilitzen, i que ja parlen de l'aplicació amb el nom comercial que li vaig posar, l'ScannerApp... Tot això em fa molta il·lusió. Segurament per a molts pot ser una cosa sense importància, però a mi em va fer il·lusió quan vaig integrar el sistema de monitoratge i errors de l'ScannerApp, al rebre la primera notificació de l'empresa client, saber que l'estaven fent servir em va donar una satisfacció molt gran.

Apuntar que a GISCE, hi ha hagut molt bon ambient de treball, i poder desenvolupar el projecte al costat d'ells m'ha ajudat molt a créixer com a programador i com a persona. A més, l'empresa ha decidit contractar-me, i poder continuar així amb l'aplicació a més d'altres tasques per mantenir l'ERP.

Durant el procés de desenvolupament, s'ha buscat software que utilitzés els mateixos processos o les mateixes directrius que aquest, però no s'ha trobat res similar. Això fa que haver assolit els objectius marcats en un inici del projecte, juntament amb haver aconseguit un software que no és una còpia o imitació de ningú, donin un plus extra a l'aplicació i a l'empresa.

Ha estat molt necessari l'estudi del protocol de twain, i moltes de les hores invertides han passat per haver d'aprendre a treballar amb aquest protocol. Si el fet d'haver realitzat aquest projecte, pot també ajudar a algú a abstreure's d'implementar els algorismes ja realitzats per interactuar amb el protocol, millor que millor.

A nivell d'aprenentatge, el TFG m'ha aportat diverses coses, més de les que realment m'esperava, i també fa certa il·lusió mencionar que s'han tocat molts i molts aspectes apresos durant la carrera: enginyeria del software, base de dades, sockets, threading, xarxes, APIs, control de versions, graphs d'estats... En resum, molts aspectes. És satisfactori cloure una etapa conjuntant tota una sèrie de coneixements dispersos a l'inici, i clars al final. En definitiva el TFG m'ha ajudat a aprendre molt millor i veure més clar moltes coses de la carrera.

S'ha pogut treballar amb moltes eines i metodologies que no havia utilitzat mai, fins i tot al conèixer eines com PyQt m'han animat a iniciar nous projectes. Actualment ja estic treballant amb un projecte paral·lel gràcies a conèixer diverses eines al fer aquest TFG. He pogut també aprendre a un nivell molt més aprofundit el llenguatge Python, que no havia tocat molt poc, i he gaudit molt amb aquest llenguatge, ja que m'ha fet la vida més fàcil en el projecte, i comparant-ho amb d'altres... bé, potser encara hi seríem.

Cal dir que compilar i treballar directament a nivell de controladors pot ser una mica complicat, i a voltes desenvolupar entre arquitectures de sistema, diferents versions de protocols com twain, i treballar més a baix nivell pot ser una mica complicat, però això encara m'ha motivat més aportant-me satisfacció i coneixements cada cop que trobava una solució.

Aprofito aquestes línies per fer menció i donar les gràcies a TOTS els companys de l'empresa GISCE, que en tot moment que he necessitat un cop de mà, me l'han donat d'una forma molt professional i eficient.

Subjectivament penso que l'ERP de l'empresa guanya amb aquest projecte una funcionalitat força gran. De fet ja són varis els clients que estan assabentats de la incorporació de l'ScannerApp, perquè realment els aporta una comoditat i automatització molt bona. I cal senyalar que el producte ha assolit l'objectiu de solucionar un gran problema que es plantejava: l'ScannerApp abstreu totalment al client de: si ha barrejat documents, o no sap on els ha desat, o si els adjunta i després els vol previsualitzar, o si ha de repetir el document i després no sap quin és el bo, o si ha d'anar a la carpeta a eliminar-lo, o si ha de canviar el nom del document, o de si n'ha adjuntat uns quants i ha perdut el recurs on treballava, o de si cada cop ha de seleccionar el mateix escàner i les preferències de digitalització, o que cada controlador de l'escàner sigui diferent, o si... De tot això ja no se n'ha de preocupar.

Anotar que el producte realitzat en aquest TFG, l'ScannerApp, ja està actualment desplegat, a l'empresa client. Finalment dir que ha estat un projecte que ha complert els objectius, que m'ha fet créixer a nivell personal i a nivell d'aprenentatge, i que sobretot ha reafirmat que m'apassiona la informàtica.

# CAPÍTOL 13

---

## Bibliografia

---

- Python Software Foundation. (2009). Usando el intérprete de Python. *Tutorial de Python (y Django!)*. Recuperat de: <http://docs.python.org.ar/tutorial/2/interpreter.html> [Gener 2017]
- Senkom, D. (2013). pytwain. *pytwain*. Recuperat de: <https://github.com/denisenkom/pytwain> [Gener 2017]
- Colonel, P. (2012). Pip in Windows. *StackOverflow*. Recuperat de: <http://stackoverflow.com/questions/4750806/how-do-i-install-pip-on-windows> [Gener 2017]
- TWAIN WORKING GROUP. (2015). TWAIN Linking Images with Applications. *TWAIN Linking Images with Applications..* Recuperat de: <http://www.twain.org> [Gener 2017]
- Python Software Foundation. (2007). Python 3.6.1 documentation. *Python Documentation*. Recuperat de: <https://docs.python.org/3/> [Gener 2017]
- TWAIN WORKING GROUP. (2015). TWAIN Specification Discussion. *TWAIN Linking Images with Applications..* Recuperat de: <http://www.twain.org/forums/forum/twain-specification/twain-specification-discussion/> [Gener 2017]
- Senkom, D. (2013). Pytwain's documentation. *Twain Module*. Recuperat de: <http://pytwain.readthedocs.io/en/latest/twain.html> [Febrer 2017]
- Granger, B. y Ragan-Kelley, M. (2011). The PyZMQ API. *PyZMQ Documentation*. Recuperat de: <https://pyzmq.readthedocs.io/en/latest/api/zmq.html> [Febrer 2017]
- iMatix Corporation. (2007). Get The Software. *Distributed Messaging*. Recuperat de: <http://zeromq.org/intro:get-the-software> [Febrer 2017]
- Bodnar, J. (2007). Events and signals in PyQt5. *ZetCode brings tutorials for programmers in various areas*. Recuperat de: <http://zetcode.com/gui/pyqt5/dialogs/> [Febrer 2017]
- García, L. (2015). Un poco de ZeroMQ. *Un poco de Java*. Recuperat de: <https://unpocodejava.wordpress.com/2013/02/25/un-poco-de-zeromq/> [Febrer 2017]

- Riverbank Computing Ltd, Qt Company. (2015). QComboBox Class Reference [QtGui module]. *PyQt4 Reference Guide*. Recuperat de: <http://pyqt.sourceforge.net/Docs/PyQt4/qcombobox.html> [Febrer 2017]
- Granger, B. y Ragan-Kelley, M. (2011). zmq Python bindings for 0MQ. *PyZMQ Documentation*. Recuperat de: <https://pymq.readthedocs.io/en/latest/api/index.html> [Març 2017]
- Bodnar, J. (2007). Dialogs in PyQt5. *ZetCode brings tutorials for programmers in various areas*. Recuperat de: <http://zetcode.com/gui/pyqt5/eventssignals/> [Març 2017]
- Lundh, F. (2005). The Python getattr Function. *effbot.org*. Recuperat de: <http://effbot.org/zone/python-getattr.htm> [Març 2017]
- Qt Company. (2015). Creating Main Windows in Qt Designer. *Qt Documentation*. Recuperat de: <http://doc.qt.io/qt-4.8/designer-creating-mainwindows.html> [Març 2017]
- Fred, H. (2014). TWAIN sample Data Source and Application. *TWAIN Data Source [DS]*. Recuperat de: <https://sourceforge.net/projects/twain-samples/files/TWAIN%202%20Sample%20Data%20Source/TWAIN%20DS%202.1.3/> [Març 2017]
- Dynamsoft Corporation. (2014). ErrorCode Enumeration. *Dynamsoft.com*. Recuperat de: [http://www.dynamsoft.com/help/TWAIN/.Net-TWAIN-Scanner/html/T\\_Dynamsoft\\_DotNet\\_TWAIN\\_Enums\\_ErrorCode.htm](http://www.dynamsoft.com/help/TWAIN/.Net-TWAIN-Scanner/html/T_Dynamsoft_DotNet_TWAIN_Enums_ErrorCode.htm) [Març 2017]
- Qt Company. (2015). Qt Widgets C++ Classes. *qt.io*. <http://doc.qt.io/qt-5/qtwidgets-module.html> [Març 2017]
- Min, R. (2011). Changes in PyZMQ. *PyZMQ Documentation*. Recuperat de: [http://pymq.readthedocs.io/en/latest/changelog.html](https://pymq.readthedocs.io/en/latest/changelog.html) [Abril 2017]
- García, L. (2015). ZeroMQ (0MQ): Mensajería multilenguaje brokerless. *Un poco de Java*. Recuperat de: <https://unpocodejava.wordpress.com/2010/12/07/zeromq-0mq-mensajeria-multilenguaje-brokerless/> [Abril 2017]
- Brandl, G. (2007). First Steps with Sphinx. *SPHINX*. Recuperat de: <http://www.sphinx-doc.org/en/stable/tutorial.html> [Abril 2017]
- Senkom, D (2015). Mention that you depend on TWAINDSM #4. *GitHub*. Recuperat de: <https://github.com/denisenkom/pytwain/issues/4> [Abril 2017]
- Gill, K. (2007). General Exceptions Generated by twainmodule. *twainmodule: A TWAIN Interface for Python*. Recuperat de: <http://twainmodule.sourceforge.net/docs/exception.html> [Abril 2017]
- Clark, A. y Lundh, F. (2007). Pillow Image Module. *Pillow*. Recuperat de: <http://pillow.readthedocs.io/en/3.0.x/reference/Image.html> [Abril 2017]
- Tuininga, A. (2015). cx\_Freeze's documentation. *cx\_Freeze*. <https://cx-freeze.readthedocs.io/en/latest/> [Abril 2017]
- Martin, C. (2016). KeyError: 'TCL\_Library' when I use cx\_Freeze. *StackOverflow*. Recuperat de: <https://stackoverflow.com/questions/35533803/keyerror-tcl-library-when-i-use-cx-freeze> [Abril 2017]
- Senkom, D (2006). How to install properly to avoid issues? #11. *GitHub*. Recuperat de: <https://github.com/denisenkom/pytwain/issues/11> [Abril 2017]

- Thoma, M. (2014). Config Parser. *Configuration files in Python*. Recuperat de: <https://martin-thoma.com/configuration-files-in-python/> [Maig 2017]
- Brandl, G. (2007). reStructuredText Primer. *SPHINX*. Recuperat de: <http://www.sphinx-doc.org/en/stable/rest.html> [Maig 2017]
- Senkom, D. (2016). Twain Module Raise an Exception : twain.excTWCC\_SEQERROR #7. *GitHub*. Recuperat de: <https://github.com/denisenkom/pytwain/issues/7> [Maig 2017]
- Anapolli, J. (2014). Use cx-freeze to create an msi that adds a shortcut to the desktop. *StackOverflow*. Recuperat de: <https://stackoverflow.com/questions/15734703/use-cx-freeze-to-create-an-msi-that-adds-a-shortcut-to-the-desktop> [Maig 2017]
- Mozilla. (2011). A general-purpose, web standards-based platform for parsing and rendering PDFs. *PDF.js*. Recuperat de: [https://mozilla.github.io/pdf.js/getting\\_started/#download](https://mozilla.github.io/pdf.js/getting_started/#download) [Juny 2017]
- Klein, B. (2011). Python Advanced: Threads and Threading *Python-course*. Recuperat de: <http://www.python-course.eu/threads.php> [Juny 2017]
- Fernández, A. (2012). *Python 3 al descuberto*. Madrid. Espanya. RC Libros