Universitat de Girona
Escola Politècnica Superior

Projecte fi de grau

Estudi: Grau en Enginyeria Informàtica

Títol: Disseny i desenvolupament d'una eina per a l'edició de plantilles Mako: UI-QMako

Document: Memòria

Alumna: Mª del Mar Jené Oliveras

Tutor: Jordi Regincós Isern
Departament: Informàtica, matemàtica aplicada i estadística
Àrea: Llenguatges i Sistemes Informàtics

Convocatòria (mes/any): Setembre 2021

Universitat de Girona
Escola Politècnica Superior

Projecte Fi de Grau

Disseny i desenvolupament d'una eina per a l'edició de plantilles Mako: UI-QMako

Autora:
Mª del Mar Jené Oliveras

Setembre 2021

Grau en Enginyeria Informàtica

Tutor:
Jordi Regincós Isern

Agraïments

Vull agrair a tots els membres de l'equip IT de Som Energia el seu suport i acompanyament durant la realització del projecte, especialment els membres de l'equip ERP pel suport i per ajudar-me a detectar quines funcionalitats eren necessàries i quines no eren tant importants. També vull agrair a en Juan Pedro Sánchez l'orientació en com desenvolupar l'API i implementar els tests i a en David Riera l'ajuda en la implementació de la interfície web.

Finalment, també vull agrair el suport i seguiment del projecte per part del meu tutor, Jordi Regincós, al llarg del desenvolupament del treball.

Índex

Capítol 1

Introducció

Aquest Projecte de Final de Grau s'ha desenvolupat a la cooperativa Som Energia, una comercialitzadora i productora d'energies renovables amb seu a Girona.

1.1 Som Energia

Som Energia és una cooperativa sense ànim de lucre que desenvolupa dues activitats: la producció i la comercialització d'energia de fonts 100% renovables.

La cooperativa es va fundar a finals de l'any 2010, com un projecte de participació ciutadana per a canviar el model energètic, i durant l'any 2011 va començar a comercialitzar electricitat i a impulsar els primers projectes de generació d'energia renovable.

Actualment té més de 70.000 socis, més de 130.000 contractes i té prop de 100 treballadors.

Comunicacions de Som Energia

Som Energia no té oficines d'atenció al públic, i qualsevol consulta, tràmit o modificació es realitza en línia. La confirmació o justificant de les peticions rebudes s'envia als clients o socis per correu electrònic. També es comuniquen per correu les factures o canvis de preus de les tarifes, per exemple.

Aquests correus electrònics són comunicacions automatitzades, i per gestionar-ne l'enviament, Som Energia utilitza el mecanisme que té integrat el seu ERP (programa de gestió de recursos).

Som Energia treballa amb *OpenERP*, un software de codi obert escrit en Python. Aquest programa utilitza la llibreria de plantilles Mako [Bayer 2021a] tant per definir els correus electrònics com els adjunts que aquests puguin contenir.
La llibreria Mako permet, utilitzant sintaxi semblant a HTML i a Python, definir el text i l'estil de la comunicació, permetent l'ús de variables i fragments de codi Python.

1.2 Motivació del Projecte

Actualment, la creació i modificació de les plantilles Mako de les comunicacions recauen en l'equip IT per diferents motius.

El primer motiu és la presència de codi HTML i de Python a les plantilles, que fa necessaris coneixements de programació.

El segon motiu és la voluntat de tenir un sistema de control de versions sobre el contingut de les plantilles. Actualment s'utilitza GIT i per tant es fan necessaris coneixements d'aquest software.

Finalment, en tractar-se de comunicacions massives, hi ha la voluntat d'assegurar que els canvis que es fan són correctes amb diferents casuístiques (contractes amb diferents tarifes, diferents idiomes de la comunicació...). Per això, cal comprovar el renderitzat resultant amb diferents casos, tants com sigui possible.
Per a realitzar aquest testeig, el departament IT de Som Energia va desenvolupar una eina que permet executar tests que comparen el resultat de la renderització de la plantilla amb un cas abans i després dels canvis. Aquesta eina s'executa per comandes i requereix tenir un servidor d'ERP executant-se en local i una Base de Dades que sigui una còpia recent de la de producció, també en local.
Aquests requeriments tan exigents fan que només la puguin usar membres de l'equip IT que han fet la instal·lació en el seu ordinador, i comporta un problema de seguretat en cas de pèrdua o robatori de les màquines. A més, el requisit de la mida de la memòria que han de tenir els equips cada vegada és més gran, i aviat serà inassumible.

Amb aquest projecte es vol aconseguir que les persones que defineixen els texts i els estils de les comunicacions siguin les que puguin crear i modificar les plantilles, i que només calgui intervenció de l'equip IT quan calgui modificar-ne el codi Python.

1.3 Objectius del Projecte

L'objectiu d'aquest projecte és l'anàlisi, disseny i implementació d'una eina per a l'edició de plantilles Mako a Som Energia.

Aquesta eina ha de tenir una interfície d'usuari que permeti editar les plantilles desacoblant les tres parts diferenciades: text, disseny i codi, de manera que cada perfil d'usuari pugui editar autònomament la part per la qual té coneixements.

Les edicions han de partir sempre de l'última versió de la plantilla i l'eina ha d'incorporar un sistema de control de versions per a les versions definitives.

També ha d'incorporar un sistema de testeig, de manera que permeti visualitzar el resultat de renderitzar una edició amb les dades d'un cas real.

A més, l'eina ha de permetre l'actualització de la plantilla a l'ERP amb els canvis realitzats.

1.4 Estructura del document

Els apartats del document, amb una breu explicació sobre el seu contingut, són els següents:

- Introducció: contextualització de l'empresa on es duu a terme el projecte i recull de la motivació i objectius del projecte.
- Estudi de la viabilitat: s'analitza la viabilitat del projecte a nivell econòmic, tecnològic, legal i operacional.
- Metodologia: detall sobre les metodologies de planificació i de desenvolupament seguides.
- Marc de treball i conceptes previs. Conté una explicació sobre les plantilles Mako, el seu ús a Som Energia, antecedents de l'eina a desenvolupar i una breu explicació de la metodologia de desenvolupament utilitzada.
- Requisits del sistema: recull dels requisits funcionals i no funcionals de l'eina a desenvolupar.

- Planificació: detall de la planificació del projecte.
- Estudi i decisions: recull i justificació de les decisions preses durant el disseny de l'aplicació.
- Anàlisi i disseny del sistema: estudi de les necessitats del sistema i explicació del disseny proposat.
- Implementació i proves: explicació de la preparació de l'entorn de desenvolupament i dels reptes solucionats durant la realització del projecte. També inclou un recull de les proves realitzades.
- Implementació i resultats: es mostren els objectius assolits i el funcionament de l'eina desenvolupada.
- Conclusions: discussió sobre l'eina implementada i el grau d'assoliment dels objectius.
- Treball futur: recull de futures ampliacions i treballs sobre l'eina desenvolupada.
- Annexos: recull de les tasques de la planificació i documentació de l'API.

Capítol 2

Estudi de Viabilitat

Aquest projecte pretén facilitar una tasca que ja s'està realitzant a Som Energia: la modificació i creació de plantilles per a correus electrònics.

L'estudi de viabilitat la podem desglossar diferents aspectes: viabilitat econòmica, tecnològica, legal i operacional.

2.1 Viabilitat econòmica

Pel que fa al cost de salari, cal tenir en compte el temps destinat al recull de requeriments (on hi participaré juntament amb altres persones de l'equip IT), el temps d'anàlisi, desenvolupament i documentació (només hi dedicaré temps jo) i el temps de seguiment del projecte (hi participarà una persona de l'equip IT i jo). Les hores destinades a la redacció del full de projecte i la memòria no s'han tingut en compte, ja que són necessaris en un context acadèmic, però no són propis d'un projecte de desenvolupament en el marc d'una empresa.

El temps planificat d'anàlisi, aprenentatge i desenvolupament per part meva és de 332 hores.

Pel recull de requeriments, la previsió és destinar-hi 2 h cada una de les persones involucrades.

Finalment, el temps previst pel seguiment del projecte és de 6 h.

Les hores de mentoria i suport realitzades per part de persones de l'equip IT de Som Energia són poques, i l'impacte en l'àmbit econòmic és menyspreable.

Per tant, les hores d'una persona programadora júnior (jo) seran 332+2+6 = 340 h, i les hores de programador sénior seran 3*2+6 = 12 h.

Pel que fa a llibreries utilitzades, totes són amb llicència Codi obert, per tant no ha calgut comprar cap llicència.

Per al desenvolupament s'ha usat un portàtil Slimbook amb sistema operatiu Ubuntu 18.4, que és l'ordinador que m'ha proporcionat Som Energia per a desenvolupar les meves tasques. El preu de l'aparell és de 1.200 €, i l'he utilitzat durant 340 h de desenvolupament del projecte (2,1 mesos d'ús comptant una jornada laboral de 40 h). Amortitzant-lo a 6 anys suposa un cost de 16,67 € al mes.

Finalment, Som Energia té diferents servidors per a desplegar les seves aplicacions, i no ha calgut adquirir-ne cap de nou per a poder allotjar l'UI-QMako.

Així doncs, el cost econòmic desglossat és el següent:

Concepte	**Temps**	**Cost unitari**	**Total**
Ordinador Slimbook	2,1 mesos	16,67 €/mes	35,05 €
Hores programadora júnior	340 h	9 €/h	3060 €
Hores programador sénior	12 h	15 €/h	180 €
Total			**3275,05 €**

És un cost assumible per la cooperativa, per tant el projecte és viable a nivell econòmic.

2.2 Viabilitat tecnològica

Tant el software utilitzat (sistema operatiu GNU/Linux, IDE, *frameworks* de desenvolupament...) com el hardware (ordinador, servidors) necessaris per al desenvolupament i instal·lació de l'eina són fàcilment accessibles, per tant en l'àmbit tecnològic el projecte és viable.

2.3 Viabilitat legal

El desenvolupament es realitzarà només amb eines i components de software lliure o bé pels quals Som Energia en tingui llicència d'ús.

No es recolliran dades dels usuaris que utilitzin l'eina desenvolupada (excepte un nom d'usuari i contrasenya).

Les dades utilitzades pel testeig de les plantilles són dades que les persones sòcies i clientes de la cooperativa han cedit per a la relació contractual que les uneix a la cooperativa. Aquestes dades no se cediran a tercers ni s'utilitzaran per cap altre fi que no sigui poder realitzar les comunicacions necessàries d'una manera correcta i entenedora.

Per tant, el projecte és viable a nivell legal.

2.4 Viabilitat Operacional

Abans de poder avaluar la viabilitat operacional del projecte, cal tenir present quin és el flux de treball per poder realitzar aquestes modificacions actualment:

1. L'equip responsable de la comunicació electrònica, amb el suport de Comunicació, elabora el text modificat i l'estil (HTML i CSS) que ha de tenir.
2. Aquest equip elabora (o fa una petició a les persones traductores) les traduccions.
3. Amb el text ja definit, es fa una petició a l'equip IT perquè facin les modificacions. Aquesta petició cal que sigui prioritzada entre les altres peticions que fan tots els equips de Som Energia. Si hi ha recursos d'IT disponibles, el dia d'inici del següent *Sprint*, aquesta tasca entra dins l'*Sprint* (ja que la metodologia de treball és *SCRUM*).
4. La/les persones d'IT fan la modificació, en fan el testeig i demanen a l'equip promotor que verifiqui el resultat.
5. Un cop el resultat ha estat revisat, s'aplica a l'ERP de producció els canvis realitzats.

Aquest flux de treball fa que sigui poc àgil realitzar modificacions, i allunya la tasca de l'equip que la necessita. A més, fa que es destinin recursos de IT en tasques que podrien ser realitzades o bé pel mateix equip, o bé per persones de comunicació que tenen coneixement d'HTML i CSS. Amb l'eina desenvolupada en el marc del Projecte de Final de Grau es vol aconseguir que el flux sigui més àgil i que les persones que defineixen el text i l'estil siguin les que duguin a terme les tasques.

La solució proposada aconsegueix que les persones amb el coneixement del text i disseny que han de tenir les plantilles puguin editar-les, alliberant a les persones d'IT d'aquesta tasca. També permet facilitar el flux de traducció, punt essencial per a poder traduir les comunicacions de la cooperativa als quatre idiomes oficials de l'estat (ja que cap persona de IT sap ni gallec ni basc). A més, el manteniment de l'eina és baix i fàcil de realitzar.

2.5 Valoració global de la viabilitat

El desenvolupament del projecte és viable en tots els aspectes analitzats. Gràcies a l'eina desenvolupada, es pot reduir a la meitat les hores que impliquen una modificació en text o estil de les plantilles de correu electrònic, ja que les persones que defineixen el text i/o estil poden fer-ho directament amb l'eina proposada. A més, minimitza la comunicació entre equips i el temps de gestió i planificació d'aquests.

Per tant, la valoració global és positiva.

Capítol 3

Metodologia

3.1 Metodologia de Planificació

La metodologia de desenvolupament del projecte serà la mateixa que s'utilitza dins l'equip IT de Som Energia, basada en *SCRUM Agile*. Dins la cooperativa, els *sprints* s'anomenen Rondes, i la puntuació de la complexitat de les tasques es basen en el temps que comportarà el desenvolupament d'aquella tasca. En aquest cas, les Rondes s'han establert amb una durada de dues setmanes, començant en dilluns i acabant en divendres.

Les primeres persones interessades en el desenvolupament de l'eina són les persones de l'equip ERP, dins de l'equip IT, ja que substitueix l'script que s'està utilitzant actualment. Juntament amb elles, aniré definint les històries d'usuari i tasques a desenvolupar. Aquestes tasques s'intentarà que siguin independents unes de les altres, i ben definides respecte a la descripció i l'abast. En aquesta primera etapa, el rol d'*Scrum Master* i de *Product Owner* recauen en la persona de referència del projecte de l'equip ERP. Al final de cada ronda i inici de la següent, em reuniré amb ella per una demostració de les tasques fetes, per a prioritzar les noves tasques a realitzar i marcar quines entren per a desenvolupar durant les dues setmanes següents. Com a membre de l'equip ERP, cada dia participaré en una *Stand-up Meeting*, on podré comentar l'estat del projecte i comentar els problemes o dubtes que sorgeixin durant el desenvolupament.

Quan l'eina estigui més desenvolupada, caldrà incorporar la visió de noves usuàries: persones de l'equip que no són d'IT (per exemple, de l'equip d'Atenció i Suport, o Factura). En aquesta segona etapa la funció del *Product Owner* la realitzarà una persona externa a IT que reculli les necessitats que caldrà implementar.

3.2 Metodologia de desenvolupament

Durant el desenvolupament del projecte s'utilitzarà un Sistema de Control de Versions, concretament git, i els repositoris s'allotjaran a la plataforma GitHub.

Sempre que sigui possible, la part del backend del Projecte s'intentarà realitzar amb Test-Driven Development (TDD), per aconseguir tenir una cobertura de tests elevada, i aconseguir una millor qualitat de codi. Per la part del frontend també es podrien desenvolupar tests, però Som Energia no té prou experiència en testejar interfícies web, i per això el testeig serà manual.

Tant les metodologies de desenvolupament com de planificació escollides permeten tenir agilitat en el desenvolupament, adaptabilitat als canvis i reduir el temps necessari per a poder desplegar noves funcionalitats. El fet de treballar amb control de versions permet tenir control sobre els canvis que s'incorporen, i poder revertir possibles errors que es detectin amb facilitat. També és una eina important en projectes col·laboratius, tot i que el desenvolupament el realitzaré només jo fins a tancar la versió final pel projecte.

Capítol 4

Marc de treball i conceptes previs

En aquest apartat s'explica la sintaxi de les plantilles Mako, quin ús se'n fa a Som Energia i el projecte precursor de l'eina desenvolupada en aquest projecte. També s'explica breument la metodologia TDD seguida durant el desenvolupament del backend.

4.1 Plantilles Mako

La llibreria de plantilles Mako [Bayer 2021a] està escrita en Python i és semblant a altres llibreries usades habitualment per a aquesta finalitat, com poden ser les plantilles de Django [Foundation 2021] o Jinja2 [Pallets 2007b]. Està publicada sota una llicència MIT.

Aquesta llibreria està incorporada en l'ERP utilitzat a Som Energia (OpenERP) i renderitzar plantilles definides en HTML i CSS, i que incorporen codi i variables amb sintaxi semblant a Python.

Un exemple bàsic de l'ús d'aquesta llibreria és el següent:

```
from mako.template import Template

mytemplate = Template("<p>Hola, ${name}!</p>")
print(mytemplate.render(name="món"))
```

Listing 1: Exemple bàsic d'ús de la llibreria Mako

El resultat d'aquest petit fragment de codi és:

```
<p>Hola, món!</p>
```

Listing 2: Resultat de l'execució de l'exemple anterior

4.1.1 Sintaxi

Substitució d'expressions

Amb la construcció *${expressio}*, es poden representar expressions vàlides en Python, siguin variables simples o bé altres expressions complexes. Normalment les variables venen definides pel context passat en la funció de renderització, o bé poden haver estat definides en algun altre punt previ de la plantilla.

Estructures de control

Es poden utilitzar estructures de control, com són els blocs *if/else*, *while*, *for*, *try/except* usant el símbol % seguit de l'expressió de Python per a aquestes estructures. Per indicar el final d'aquestes estructures, cal utilitzar altre cop el símbol % seguit de "end*nom*", on *nom* és la paraula clau de l'expressió. A més, aquestes estructures es poden combinar entre elles. Per exemple:

```
% for a in ['one', 'two', 'three', 'four', 'five']:
    % if a[0] == 't':
    its two or three
    % elif a[0] == 'f':
    four/five
    % else:
    one
    % endif
% endfor
```

Listing 3: Exemple d'ús d'estructures de control en plantilles Mako. Exemple extret de la documentació de la lliberia [Bayer 2021b].

Blocs de Python

Es poden utilitzar estructures de control, com són els blocs if/else, while, for, try/except usant el símbol % seguit de l'expressió de Python per a aquestes estructures. Per indicar el final d'aquestes estructures, cal utilitzar altre cop el símbol % seguit de "end*nom*", on *nom* és la paraula clau de l'expressió. A més, aquestes estructures es poden combinar entre elles. Per exemple:

```
<%
    from datetime import datetime
    ara = datetime.now()

    def laMevaFuncio(text):
        return "{}: text".format(ara)

    laMevaVariable = "Hola, món!"
%>
```

Listing 4: Exemple de definició d'un bloc de codi Python dins d'una plantilla.

Tags

Mako ofereix altres funcionalitats en forma de *tags*, que segueixen la sintaxi <%*tag arguments*/> o bé <%*tag arguments*> </%*tag*>. Algunes d'aquestes etiquetes disponibles són *include*, per a incloure el text d'un fitxer indicat als arguments o *def* per a definir funcions.

```
<%include file="main.html"/>

<%def name="laMevaFuncio(text)">
    el text és ${text}
</%def>
```

Listing 5: Exemple d'ús de les etiquetes *include* i *def*.

Comentaris

Els comentaris d'una sola línia es defineixen utilitzant ##, i els comentaris multilínia es defineixen entre les etiquetes *<doc>* i *</%doc>*

4.1.2 Ús de les plantilles Mako a Som Energia

Les plantilles Mako a Som Energia s'utilitzen principalment per a definir dos tipus de comunicacions: correus electrònics i fitxers adjunts.

El model que defineix els correus electrònics dins de l'ERP és el model '*poweremail.templates*', que conté el text de la plantilla Mako, l'assumpte del correu, destinataris, destinataris en còpia (tots ells definits també utilitzant sintaxi Mako), i altres camps com poden ser el compte de correu des d'on s'envia, el nom de la plantilla... Les instàncies d'aquest model estan guardades a la Base de Dades (PostgresSQL) de l'ERP. Els models i funcions per a l'enviament de correus a l'ERP estan definits al mòdul anomenat *Poweremail*.

En canvi, la definició dels arxius adjunts ve definida pel model '*ir.xml.report*', que defineix la ruta on es troba la plantilla a utilitzar (fitxer), nom de l'adjunt, i altres variables com són els marges que han de tenir les pàgines del PDF resultant.

Tant les instàncies de '*poweremail.template*' com les de '*ir.action.xml.report*' es poden definir en *records* en fitxers XML perquè qualsevol instal·lació posterior tingui definides aquestes instàncies. En definir aquests registres, les instàncies queden associades a un XML ID (ID semàntic), al qual es pot fer referència. Aquest id semàntic és el mateix en qualsevol instal·lació de l'ERP, fet que no passa amb els IDs de la Base de Dades, ja que depenen de l'ordre de creació de les diferents instàncies. Sempre que sigui possible, es prioritzarà identificar els objectes de l'ERP mitjançant XML IDs.

Tot i que tant els correus electrònics com els documents adjunts estan definits en plantilles Mako, els models que les utilitzen es defineixen i s'emmagatzemen diferent. Això fa que les operacions de lectura i escriptura, funcionalitats clau de l'eina a desenvolupar, siguin completament diferents.

4.2 Repositori Oomakotest

L'any 2015 els membres de l'equip IT de Som Energia van iniciar el projecte Oomakotest [Som Energia 2021] per a poder gestionar els canvis realitzats a les plantilles Mako utilitzades. Aquest projecte consisteix en un programa escrit en Python, executat per línia de comandes, que permet fer diferents accions sobre les plantilles registrades:

- Llistar les plantilles disponibles
- Descarregar la versió actual d'una plantilla concreta
- Realitzar tests sobre una plantilla executar tests Back To Back (tests que comparen el resultat obtingut respecte a un resultat esperat). Cal que l'usuari executi aquesta comanda abans i després de cada edició, i el programa indica si hi ha hagut diferències. En cas que hi siguin, l'usuari ha de decidir si accepta el nou resultat com a correcte o manté el vell, per a les noves execucions.
- Pujar a producció la nova versió de la plantilla.

A més, també conté scripts de Bash per a poder realitzar comparacions entre PDF's (utilitzat pels tests Back To Back dels fitxers adjunts).

Les plantilles i els ids dels casos (objectes de l'ERP) que s'utilitzen per a fer el testeig estan definides en un fitxer amb format yaml del repositori. Aquest mateix repositori s'utilitza per a tenir un control de versions de les plantilles utilitzades.

Els requisits per a poder utilitzar l'Oomakotest, a part de la seva instal·lació, són els següents:

- Sistema Operatiu Linux
- Instal·lació completa de OpenERP, incloent una Base de Dades que sigui una rèplica tan fidel com sigui possible de la Base de Dades de producció. Per a l'execució del programa cal tenir un servidor executant-se en local.
- Tenir instal·lat un editor de text pla.
- Tenir coneixements de Bash, Git, HTML, CSS, Python i Mako.

Per aquests motius, és una eina que només pot utilitzar l'equip IT. A més, tenir rèpliques de dades reals en les màquines dels membres de l'equip suposa un risc (en cas de pèrdua, robatori o accés indegut a aquestes màquines), un temps elevat de manteniment pels canvis en el codi, i fixa uns requeriments de memòria pels ordinadors utilitzats molt exigents.

L'eina desenvolupada durant el transcurs d'aquest Projecte pretén substituir el repositori Oomakotest, evitant els requeriments prohibitius. A més, pretén facilitar-ne l'ús per persones no programadores mitjançant la incorporació d'un editor HTML WYSIWYG (*What You See Is What You Get*).

4.3 Test-Driven Development

El desenvolupament guiat per tests (TDD en les seves sigles en anglès) és una metodologia de programació desenvolupada per Kent Beck [Beck 2002], i té relació amb els conceptes de primer-el-test de extreme programming. Els passos del TDD són els següents:

1. Afegir un test: programar un test per a una funcionalitat concreta i un cas concret.
2. Executar tots els tests: tots els testos han de passar bé, excepte l'últim que hem escrit, ja que encara no s'ha programat aquesta funcionalitat i casuística. Si el test no falla, o bé ja teníem el codi per aquest cas o bé el test està malament.
3. Escriu el mínim codi possible per a que el nou test passi.
4. Executar els tests i ara tots haurien de ser positius.
5. Refactoritzar el codi per eliminar codi duplicat, fer-lo més llegible o fer-lo més fàcil de mantenir. Cal executar els tests després de qualsevol canvi per assegurar que no s'han introduït errors durant el procés.

Repetir els passos per cada nova funcionalitat. Els tests haurien de ser petits i incrementals, i cal mantenir-los tots amb un resultat satisfactori per cada desenvolupament que es fa.

Capítol 5

Requisits del sistema

L'anàlisi de requisits del sistema es va realitzar amb membres de l'equip IT de Som Energia, ja que aquest projecte va dirigit a desenvolupar una eina que s'utilitzarà en el dia a dia de la cooperativa.

Per tenir la visió de tots els tipus d'usuari que utilitzaran l'eina hagués estat interessant que futures persones usuàries que no formin part de l'equip IT també haguessin participat en el recull de requisits. Això no ha estat possible degut a la càrrega de feina de l'equip, agreujada per l'entrada en vigor de noves lleis del mercat elèctric, i queda pendent per les següents fases de desenvolupament.

La majoria de requeriments, tant funcionals com no funcionals, han vingut marcats per la cooperativa.

Tenint en compte el temps disponible per a la realització del projecte, i que moltes de les funcionalitats són comunes, el desenvolupament es centra en les plantilles de correus electrònics i no en les plantilles de documents adjunts.

5.1 Usuaris

Abans de decidir els requisits de l'eina, cal identificar els usuaris a qui va dirigit. Els diferents tipus d'usuaris que s'han detectat són els següents:

- Persones amb coneixements de Python, HTML i CSS: membres de l'equip IT.
- Persones amb coneixements d'HTML i CSS: membres de l'equip de comunicació.
- Persones sense coneixements d'HTML ni CSS: membres dels altres equips de Som Energia.

- Persones traductores a altres idiomes: no tenen per què tenir coneixements de programació.

5.2 Requeriments funcionals

Un cop analitzats els diferents tipus d'usuaris, s'ha fet l'anàlisi de requeriments de l'eina a desenvolupar. Com que el projecte és molt ambiciós, s'han dividit els requeriments en grups: requisits necessaris, desitjables i futurs. En el marc del projecte es desenvoluparan els requisits necessaris i desitjables, deixant els futurs per fases posteriors.

Requisits necessaris

- Qualsevol usuari s'ha de poder registrar al sistema, i se li assignarà la categoria de permisos bàsics.
- Qualsevol usuari ha de poder veure el llistat de plantilles disponibles.
- Qualsevol usuari ha de poder afegir una plantilla (de les existents a l'ERP).
- Qualsevol usuari ha de poder veure la versió actual d'una plantilla.
- Qualsevol usuari amb permisos bàsics ha de poder editar el contingut (text i estil) d'una plantilla.
- Qualsevol usuari amb permisos d'edició de Python ha de poder editar el contingut i el codi d'una plantilla.
- Qualsevol usuari amb un permís d'edició ha de poder guardar els canvis realitzats.
- Qualsevol usuari ha de poder afegir un cas de testeig d'una plantilla.
- Qualsevol usuari ha de poder veure el resultat (renderitzat) de la seva edició, amb un dels casos de la plantilla.
- Qualsevol usuari ha de poder pujar la seva edició a l'ERP.
- Qualsevol persona administradora ha de poder assignar permisos als usuaris.

- Cal incorporar un Control de Versions de les versions definitives de les plantilles.

Requisits desitjables

- Qualsevol usuari ha de poder cercar, en el llistat de plantilles, una plantilla per nom.
- Qualsevol usuari ha de poder cercar, en el llistat de plantilles, una plantilla per model al que aplica.
- Qualsevol usuari ha de poder cercar, en el llistat de plantilles, una plantilla per ID semàntic (XML ID).
- Una usuària amb permisos d'administració ha de poder deshabilitar un altre usuari.
- En pujar els canvis a l'ERP, els usuaris han de poder escollir entre l'ERP de producció i l'ERP de testing.
- En veure el resultat (renderitzat) d'una edició, els usuaris han de poder comparar-lo amb el resultat de la versió actual.
- Una usuària amb permisos avançats ha de poder eliminar una plantilla de les disponibles.
- Una usuària amb permisos avançats ha de poder eliminar un cas de testeig d'una plantilla.

Requisits futurs

- En veure el resultat (renderitzat) d'una edició, els usuaris han de poder veure les diferències entre el resultat de la seva edició i el resultat de la versió actual.
- Qualsevol usuari amb permisos de traducció ha de poder editar les traduccions d'una plantilla.
- Qualsevol usuari ha de poder veure el llistat de plantilles organitzat per carpetes, segons l'àmbit (equip) responsable de la plantilla.

5.3 Requeriments no funcionals

Com que el projecte es desenvolupa en col·laboració amb una empresa externa a la universitat, hi ha un conjunt de decisions que no he pogut prendre jo, com pot ser en quin llenguatge de programació s'implementa el projecte. Per tant, aquestes decisions són, en realitat, requeriments no funcionals que m'han vingut marcats. Són els següents:

- Codi lliure: Som Energia és una cooperativa sense ànim de lucre. El desenvolupament de software es fa, sempre que sigui possible, amb llicències de Codi lliure i utilitzant llibreries i altres eines amb llicències d'aquest tipus.
- Llenguatge Python per a desenvolupar el backend: les aplicacions desenvolupades a Som Energia, excepte les aplicacions web, es desenvolupen en Python. Com que l'eina desenvolupada durant aquest TFG s'incorporarà a les aplicacions de la cooperativa, i l'haurà de mantenir l'equip IT d'aquesta, se'm va demanar que fos programada amb Python.
- Framework React per a desenvolupar el frontend: les aplicacions web de Som Energia s'estan migrant des d'altres frameworks JavaScript (Angular, Mithril...) cap a React, per a unificar tecnologies. Per seguir amb aquesta unificació, el frontend se'm va demanar que fos en React.
- Allotjament en servidors amb Sistema Operatiu Linux: l'eina desenvolupada ha de poder ser allotjada en els servidors que Som Energia té contractats.
- Sistema de gestió de base de dades PostgreSQL, ja que és el sistema més utilitzat a Som Energia i el coneixement d'aquest és ampli dins l'equip.

Els altres requisits no funcionals de l'aplicació són:

- La interfície d'usuari ha de ser intuïtiva i fàcil d'aprendre: una persona usuària ha de poder aprendre'n totes les funcionalitats en menys de 2 hores.
- La interfície d'usuari ha de ser en català.
- La interfície serà desenvolupada per a poder-se usar correctament en els navegadors Firefox i Google Chrome.

- En cas d'error, tant del sistema com de la persona usuària, s'ha de notificar amb un missatge descriptiu.
- El codi Python ha d'estar testejat amb tests unitaris.
- L'eina desenvolupada no ha de tenir afectació en els altres usos que té l'ERP.
- En cas de fallada, s'arxivaran logs del procés per a poder-se analitzar posteriorment.
- El mecanisme d'autenticació dels usuaris ha de ser segur.

Capítol 6

Planificació

La planificació del projecte està fortament lligada a la metodologia de desenvolupament, que és una adaptació de *SCRUM Agile*. Amb aquesta metodologia, en cada *sprint* es realitzen tasques de recull de requisits, d'anàlisi, d'implementació i de testeig.

Per a la realització del Projecte s'ha planificat un total de 5 *sprints* (anomenades Rondes a Som Energia).

Les rondes realitzades i les seves tasques són les següents:

- Ronda 0: Anàlisi i Requeriments
  - **Descripció:** anàlisi del projecte a desenvolupar i recull de requeriments generals.
  - **Tasques:**
    - Recull de requeriments i anàlisi de la interacció dels diferents elements.
    - Anàlisi de diferents frameworks per desenvolupar l'API.
    - Escollir l'ORM a utilitzar.
    - Aprenentatge dels frameworks usats pel projecte.
    - Disseny del model de dades.
    - Redacció de la memòria del projecte.
  - Temps: 70 h
- Ronda 1: Inici del desenvolupament
  - **Descripció:** inici del desenvolupament de l'eina
  - **Tasques:**

    - Crear del repositori del frontend.
    - Crear del repositori del backend.
    - Crear de models a la base de dades.
    - Retorn (API) i visualització (UI) d'un llistat de plantilles actuals.
    - Permetre afegir plantilles a partir de XML ID d'una plantilla.
    - Inicialització dels tests de l'API.
    - Testeig.
    - Redacció de la memòria del projecte.
  - Temps: 60 h

- Ronda 2: Edició de plantilles
  - **Descripció:** implementació de la funcionalitat d'edició de plantilles
  - **Tasques:**
    - Obtenir el text i informació d'una plantilla de l'ERP.
    - Escollir editor WYSIWYG per l'edició de plantilles.
    - Crear el repositori per les plantilles.
    - Implementar edició de plantilles en text pla.
    - Implementar edició de plantilles amb editor WYSIWYG.
    - Implementar control de versions de les dades de les plantilles obtingudes de l'ERP.
    - Testeig.
    - Redacció de la memòria del projecte.
  - Temps: 70 h

- Ronda 3: Afegir autenticació d'usuaris i renderització de plantilles
  - **Descripció:** afegir l'autenticació d'usuaris
  - **Tasques:**
    - Afegir l'autenticació d'usuaris.
    - Implementar el registre.
    - Implementar la modificació de permisos dels usuaris.
    - Implementar la renderització d'una plantilla.
    - Testeig.
    - Redacció de la memòria del projecte.

  - Temps: 75 h

- Ronda 4: Casos de testeig i pujar les edicions a l'ERP
  - **Descripció:** poder testejar les edicions amb diferents casos i pujar canvis a l'ERP
  - **Tasques:**
    * Implementar afegir casos de testeig d'una plantilla.
    * Implementar pujar edicions a l'ERP.
    * Implementar la iteració pel renderitzat de tots els casos.
    * Implementar la connexió amb diferents ERP (testing i producció).
    * Testeig.
    * Redacció de la memòria del projecte.
  - Temps: 70 h

- Ronda 5: Gestió d'errors, estil de la UI i altres ajustos
  - **Descripció:** gestió dels errors i visualització d'aquests a la UI, donar estil a la UI i ajustar les implementacions fetes.
  - **Tasques:**
    * Implementar la gestió dels errors.
    * Implementar la visualització dels errors al frontend.
    * Ajustar l'estil de la UI.
    * Arreglar o ajustar les funcionalitats ja implementades. Repàs de tasques pendents.
    * Testeig.
    * Redacció de la memòria del projecte.
  - Temps: 70 h

El temps estimat per al desenvolupament del projecte són 415 h. D'aquest temps, es calcula que un 10% estarà destinat a l'anàlisi, adquisició de coneixements i redacció de la memòria, un 40% del temps estarà destinat a la implementació del backend (20% implementació i 20% testeig) i el 40% restant es destinarà a la implementació del frontend.

A l'Annex I es poden veure les tasques i històries d'usuari desenvolupades a cada Ronda, amb la dedicació real i estimada de cada una d'elles.

Capítol 7

Estudi i decisions

La primera decisió que he pres durant el desenvolupament d'aquest Projecte és separar la part de comunicació amb l'ERP, amb la Base de Dades i la lògica de les operacions de la part d'interfície d'usuari. Aquestes dues parts, respectivament, són el backend i el frontend. Aquesta separació permet desacoblar la part de visualització de la part de control i models. També aporta versatilitat a l'aplicació, ja que si en algun moment es vol canviar la visualització, el canvi només afectaria una de les dues parts.

7.1 Frameworks i tecnologies pel Backend

Com ja s'ha comentat, el llenguatge de programació utilitzat per desenvolupar el backend és Python, en la versió 3, ja que és un requeriment de l'empresa. Pel que fa al disseny, el departament d'IT només treballa amb serveis REST, i per això s'implementarà una API REST. Els beneficis, en el context del projecte, d'aquest tipus d'API són:

- Possibilitat de dissenyar una API amb moltes funcionalitats i un nombre d'endpoints reduït.
- Possibilitat d'escalar bé en cas que la càrrega de peticions sigui elevada.
- Protocol àgil, àmpliament utilitzat i de fàcil comprensió.

7.1.1 Framework per desenvolupar l'API

Per a la implementació de l'API he valorat diferents frameworks en Python: Flask [Pallets 2007a], Django REST Framework [Encode 2021a] i FastAPI [tiangolo 021].

Els pros i contres de cada un d'aquests són els següents [Sandy 2021]:

Flask

Flask és un micro framework lleuger, fàcil de configurar i flexible. És fàcil d'aprendre i permet la implementació de tests unitaris. Els contres són que no incorpora gaires utilitats, tot i que es troben mòduls externs, i que no incorpora l'asincronia a les tasques.

Django

Django és un projecte de Codi obert, que té moltes funcionalitats ja incorporades. Té incorporat el disseny Model-Vista-Controlador sobretot per aplicacions on només hi ha backend. Per desacoblar la visualització del backend es pot utilitzar el mòdul de REST.

Els contres són que no es necessita un framework amb tantes funcionalitats, ja que les necessàries per l'eina desenvolupada són molt específiques i no es troben implementacions generals per aquestes, i que l'arquitectura d'API REST no és nativa del framework.

FastAPI

FastAPI és el projecte més modern dels tres, de Codi obert i ràpid. L'estructura és semblant a la de Flask. Està construït sobre estàndards per les validacions de dades, autenticació d'usuaris... A més, incorpora suport per codi asíncron. Els contres són que el framework és més modern, i per tant la comunitat no és tan extensa, ni hi ha tantes funcionalitats desenvolupades.

Per a la implementació de l'API del projecte s'ha decidit utilitzar FastAPI, ja que és senzill d'aprendre, ràpid en les respostes i està construït sobre estàndards de la comunitat.

FastAPI és un framework ràpid per al desenvolupament d'APIs REST. És robust, fàcil i intuïtiu. Utilitza la notació de tipus de Python per a la validació de dades i

està basat en estàndards com JSON Schema [Org 2007] per a la validació de dades i OpenAPI [OpenapiInitiative 2007]. FastAPI incorpora la programació asíncrona.

Per tenir una API ben simple funcionant només caldria aquest codi:

```
from fastapi import FastAPI

app = FastAPI()


@app.get("/")
async def root():
    return {"message": "Hello World"}
```

Listing 6: Exemple bàsic d'ús de FastAPI, extret del tutorial de la seva documentació [tiangolo 2007]

I executar el servidor amb la comanda:

```
$ uvicorn main:app --reload
```

Accedint a l'adreça *http://localhost:8000* podríem veure el missatge

"{"message": "Hello World"}"

Gràcies a la integració de FastAPI amb OpenAPI, accedint a *http://localhost:8000/docs* podem veure la documentació que s'ha generat automàticament per a aquesta API tant senzilla, i fer peticions d'exemple.

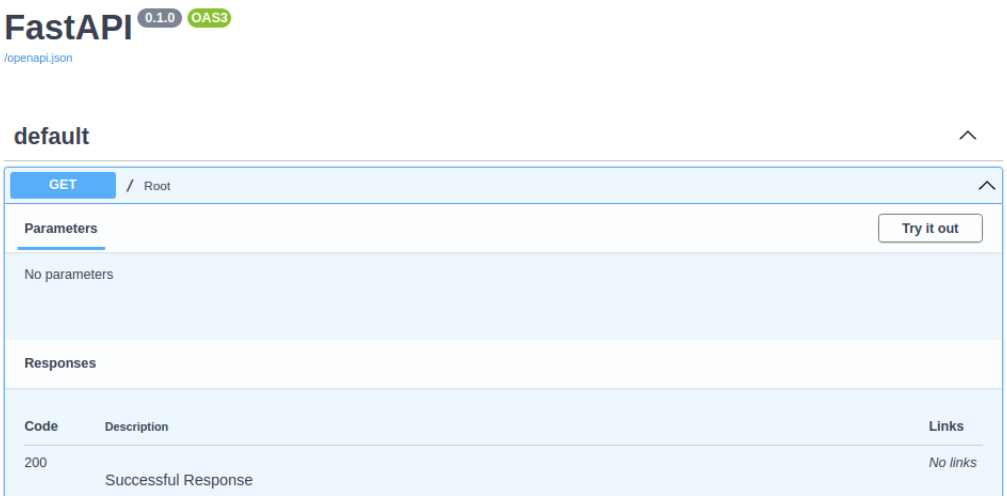

Figura 7.1: Documentació generada automàticament amb OpenApi.

Si aquest format de documentació no és el que més ens agrada, podem veure una documentació alternativa proporcionada per ReDoc accedint a la url *http://localhost:8000/redoc*.

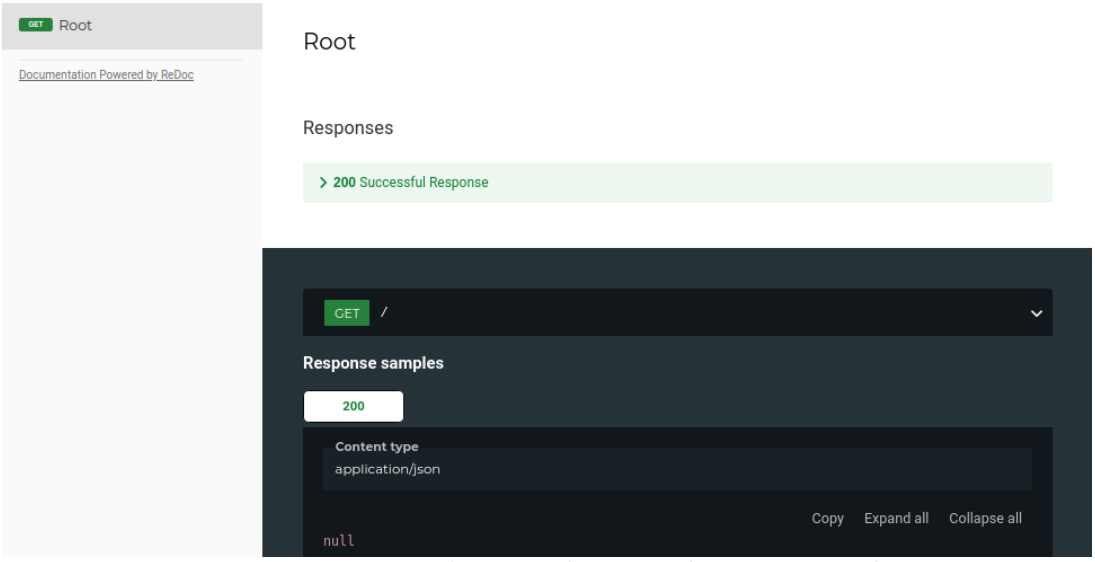

Figura 7.2: Documentació generada automàticament amb ReDoc.

Es pot consultar la documentació i les funcionalitats que proporciona el framework a la seva web oficial. [tiangolo 021].

7.1.2 ORM

Per a la interacció amb la Base de Dades des de l'API, s'ha escollit un ORM que permeti crides asíncrones, per poder aprofitar les possibilitats de FastAPI. L'ORM escollit ha estat Peewee-Async, que és una llibreria que proporciona una interfície asíncrona mitjançant *asyncio* per l'ORM Peewee [Kinev 2007], ja utilitzat a altres projectes de la cooperativa.

S'ha escollit per ser lleuger, fàcil d'utilitzar i per permetre l'asincronia en l'accés a la base de dades.

7.1.3 Pytest

FastAPI integra, gràcies a Starlette [Encode 2021b] (llibreria per construir serveis asíncrons), la classe TestClient, que permet testejar les crides a la API. A més, recomana utilitzar la llibreria Pytest per escriure i executar tests.

He decidit utilitzar Pytest [Krekel 2020b], i no la llibreria Unittest que ve incorporada a Python, per ser la recomanació dels desenvolupadors de FastAPI i per algunes de les funcionalitats que incorpora:

- És compatible amb els tests escrits en Unittest.
- És simple a l'hora d'escriure assercions.
- Permet executar tests a partir de l'últim test fallit en l'execució anterior.
- Inclou *fixtures* [Krekel 2020a], que permeten preparar conjunts de dades i objectes per a executar tests en diferents casuístiques. Permeten injectar dades i dependències als tests per facilitar escriure tests unitaris.
- Permet parametritzar tests i *fixtures*, per a evitar la duplicació de codi.

A més, Pytest és una llibreria àmpliament utilitzada i que disposa d'una bona documentació.

7.1.4 Autenticació d'usuaris

Per a la incorporació de l'autenticació dels usuaris a l'API he decidit utilitzar l'estàndard recomanat per FastAPI: OAuth2 [Parecki 2021] i l'ús de JSON Web Tokens (JWS). Tot i que aquest mecanisme permetria realitzar l'autenticació utilitzant serveis externs com poden ser GitHub o Google, de moment el servidor que emetrà el token d'autorització serà la pròpia API desenvolupada.

7.2 Frameworks i tecnologies pel Frontend

Som Energia té aplicacions web fetes amb diferents frameworks, com Angular, Mithril i React, però està migrant-les totes a aquest últim framework, per tal d'unificar tecnologies.

Per al desenvolupament d'aquest projecte, se'm va demanar que el frontend fos desenvolupat en React. Tot i no ser una decisió presa per mi, he inclòs l'apartat de React per comentar-ne les característiques i ús.

7.2.1 React

React [Facebook 2021b] és una llibreria que va ser desenvolupada inicialment per Facebook, i a partir de la seva alliberació acumula una comunitat creixent de desenvolupadors. Està focalitzat especialment al desenvolupament d'interfícies d'usuari, i *Single-page-applications*, com és el cas del frontend del projecte.

React es basa en l'ús de components, que tenen el seu propi estat, i està optimitzat per tal de rerenderitzar només els components pels quals canvia l'estat.

Un component a React ha d'implementar un mètode *render*, que retorna què cal mostrar. Aquest retorn pot ser amb sintaxi JSX, que és una sintaxi d'aparença similar a l'HTML i XML. Dins d'aquest retorn hi poden haver etiquetes HTML, altres components, o expressions avaluables en JavaScript.

El fet que els components es puguin generalitzar i reutilitzar, i que encapsulin les seves pròpies dades i funcionalitats fa que sigui fàcil estendre les aplicacions afegint nous components. Permet un desenvolupament àgil i ordenat i amb una arquitectura

fàcil de mantenir. A més, gràcies a ser una llibreria oberta amb una comunitat gran de gent que l'utilitza, hi ha disponibles eines per les parts del desenvolupament on React no arriba, com pot ser la llibreria React-Axios [sheaivey 2021] per a gestionar les peticions a APIS REST, també utilitzada en el projecte.

```
class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Hello {this.props.name}
      </div>
    );
  }
}
```

Listing 7: Exemple d'un component de React senzill, extret de la pàgina principal de React [Facebook 2021b]

7.2.2 Material-UI

Per al disseny de l'aplicació he decidit utilitzar els principis de Material Design [Google 2021]. Material és un sistema adaptable de guies, components i eines que inclouen bones pràctiques de disseny per interfícies d'usuari. Va ser creat per Google i pretén ajudar a desenvolupar interfícies de qualitat i intuïtives per aplicacions mòbil, web, o altres.

Concretament, he usat la llibreria Material-UI [mui org 2021] per React, que implementa components seguint les guies de disseny de Material Design, publicada sota una llicència MIT.

Aquesta llibreria té components ja desenvolupats, altament personalitzables, i permet un desenvolupament d'interfícies d'usuari ràpid, àgil i coherent.

7.2.3 Editor HTML WYSIWYG

Una part important del projecte és permetre que persones sense coneixements d'HTML i CSS puguin editar el text de les plantilles, de manera fàcil i intuïtiva. Per aconseguir-ho, la interfície d'usuari incorporarà un editor HTML WYSIWYG (*What you see is what you get*) per a les parts de les plantilles Mako que no siguin codi Python.

Després de buscar editors de software lliure per a React, he escollit l'editor SunEditor [JiHong88 2021], concretament el seu paquet per a React [mkhstar 2007], distribuït sota una llicència MIT. L'he escollit perquè és un editor lleuger, molt personalitzable, amb totes les funcionalitats necessàries i intuïtiu d'utilitzar.

A més, un altre dels requeriments del projecte és permetre que usuaris amb coneixement d'HTML i CSS, però sense coneixement de Python, puguin editar l'estil de les plantilles. Aquest editor permet assolir aquest requeriment, ja que permet veure i modificar el codi font del text (HTML i CSS).

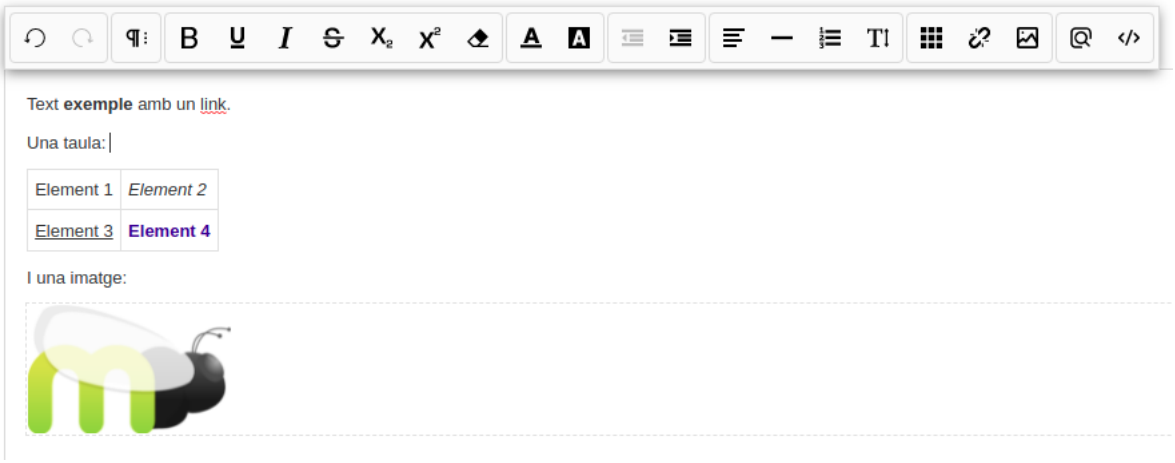

Figura 7.3: Exemple de configuració de l'editor utilitzat

7.3 Gestió de les edicions

Un dels requeriments de l'aplicació és poder incorporar un Control de Versions per a les plantilles que es posen a producció. Per a tenir aquest control de versions hi ha dues opcions: utilitzar un software de control de versions (concretament git) o bé guardar les diferents versions a la base de dades.

A part de les diferents versions de les plantilles, cal que l'eina guardi les edicions en curs dels usuaris. Una opció és usar el mateix sistema tant per les versions definitives com per les edicions en curs.

L'avantatge de desar la informació a la base de dades és que les operacions amb aquesta són fàcils de fer. El desavantatge principal és que fa difícil la detecció de canvis entre les versions.

En canvi, git té implementat la detecció de canvis i el control de versions, però per poder tenir diferents edicions en curs caldria tenir diferents branques. Pel seu funcionament, pot complicar-se gestionar els diferents estats en què es poden trobar els canvis utilitzant git, o bé resoldre possibles conflictes en les operacions, per exemple. A més, cal tenir en compte que la interacció amb repositori de git la faria l'API, que és una API REST (és a dir, sense estat). Aquesta falta d'estat de l'API faria molt complicat, per exemple, preguntar a l'usuari sobre com solucionar possibles conflictes (caldria fer passos seqüencials segons les instruccions de l'usuari), o bé que diferents usuaris fessin a la vegada diferents operacions a l'eina.

Per tal de poder aprofitar els avantatges de les diferents opcions, he decidit emmagatzemar les edicions dels usuaris a la base de dades, però aconseguir el control de versions de les versions definitives de plantilles amb git.

D'aquesta manera, el repositori per a les plantilles només tindrà una branca, que s'actualitzarà amb la versió descarregada de producció quan aquesta es consulta a l'ERP, o bé després de pujar canvis amb l'eina. En canvi, guardant les edicions en curs a la base de dades, s'aconsegueix que les operacions de lectura i escriptura d'aquestes sigui àgil. Caldrà comprovar, però, que les edicions parteixin de la versió actual de la plantilla en el moment de pujar l'edició a l'ERP (és a dir, que no s'hagin fet canvis que l'edició en curs no contingui).

Capítol 8

Anàlisi i disseny del sistema

Aquest apartat té com a objectiu analitzar els requeriments i funcionalitats de l'aplicació per a poder dissenyar l'estructura general i quina informació serà necessària guardar.

8.1 Anàlisi

Actors

Durant el recull de requeriments del sistema s'ha vist que el sistema té 3 actors definits:

- Usuaris amb coneixements de Python, HTML i CSS.
- Usuaris bàsics: podran editar la part HTML i CSS de les plantilles mitjançant un editor HTML WYSIWYG
- Administradors: podran modificar els permisos dels altres usuaris

Gràcies a la incorporació a l'eina d'un editor HTML WYSIWYG, tant les persones amb coneixement d'HTML i CSS, com les persones que no tenen aquest coneixement, podran editar còmodament el text i l'estil de la part no Python de les plantilles.

Per la metodologia iterativa de programació utilitzada (*SCRUM*), i per quedar fora de l'abast del projecte la traducció de texts, no es tenen en compte les persones traductores com a actors del sistema en l'anàlisi realitzat.

Elements del sistema

Durant el recull de requeriments i anàlisi s'ha conclòs que l'eina a desenvolupar constarà de dues parts:

- Backend (API): contindrà la lògica de l'eina, interaccionarà amb l'ERP i la Base de Dades i realitzarà el control de versions per a les plantilles.
- Frontend (UI): proporcionarà la interfície visual de l'eina i farà les peticions a l'API per a obtenir la informació necessària.

En el diagrama següent es poden veure els diferents elements i la interacció entre ells:

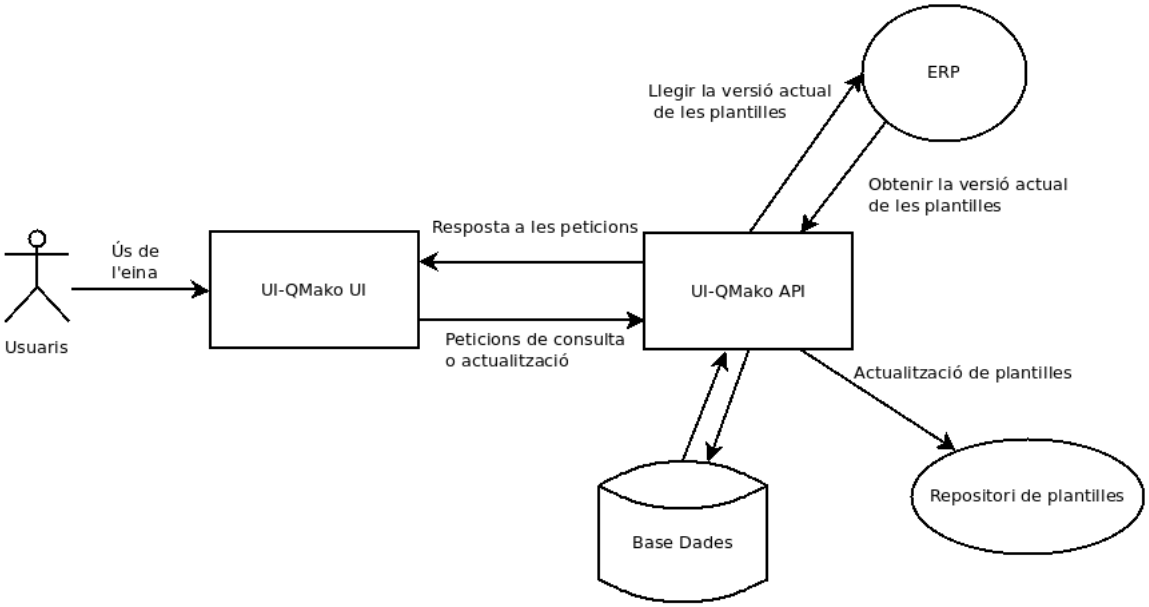

Figura 8.1: Esquema general dels elements del sistema

Casos d'ús

El diagrama de casos d'ús UML de l'aplicació és el següent:

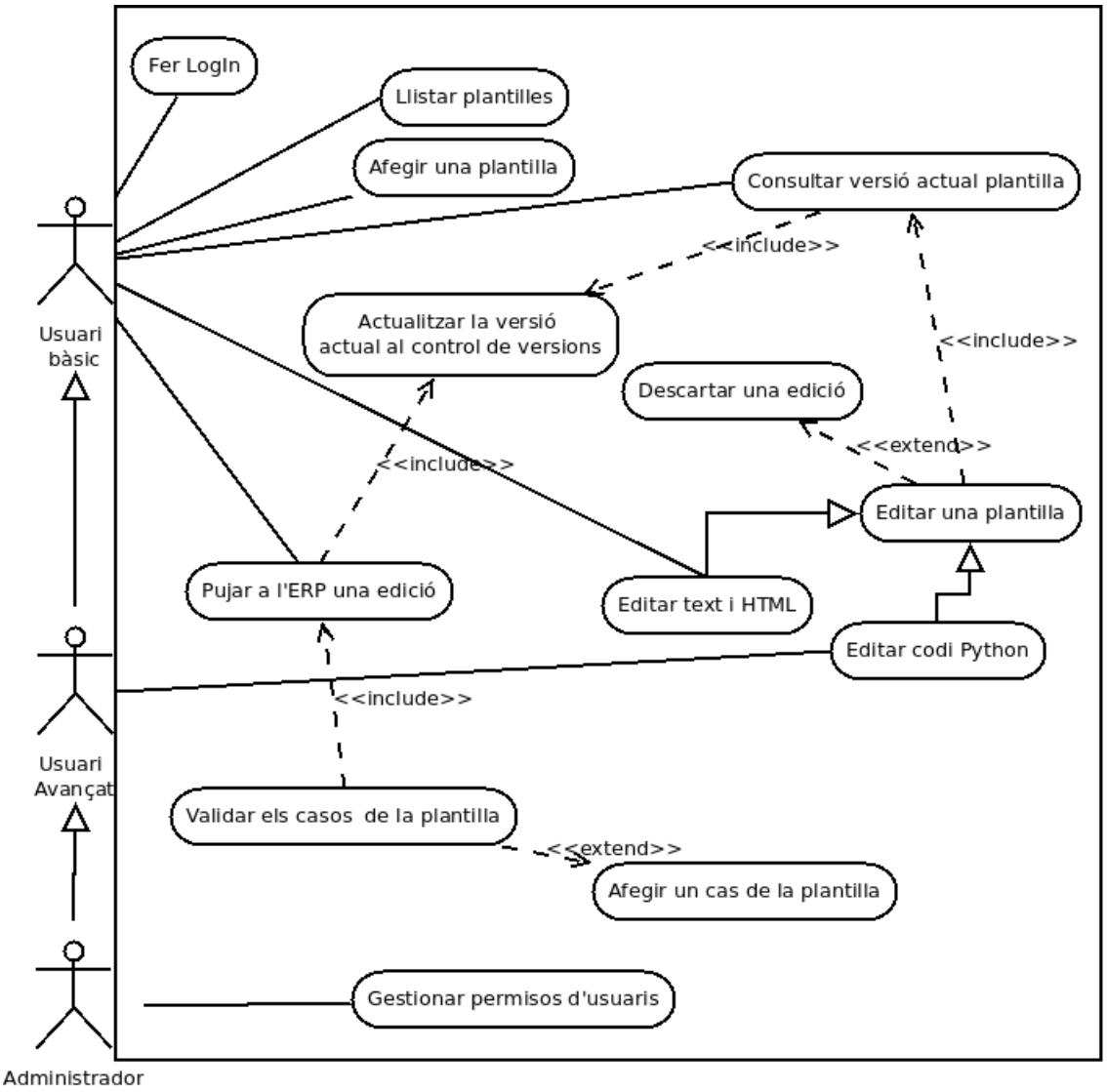

Figura 8.2: Diagrama de casos d'ús.

Model de dades

De l'anàlisi conceptual de les entitats d'informació que han de tenir persistència, s'extreu el següent model entitat-relació:

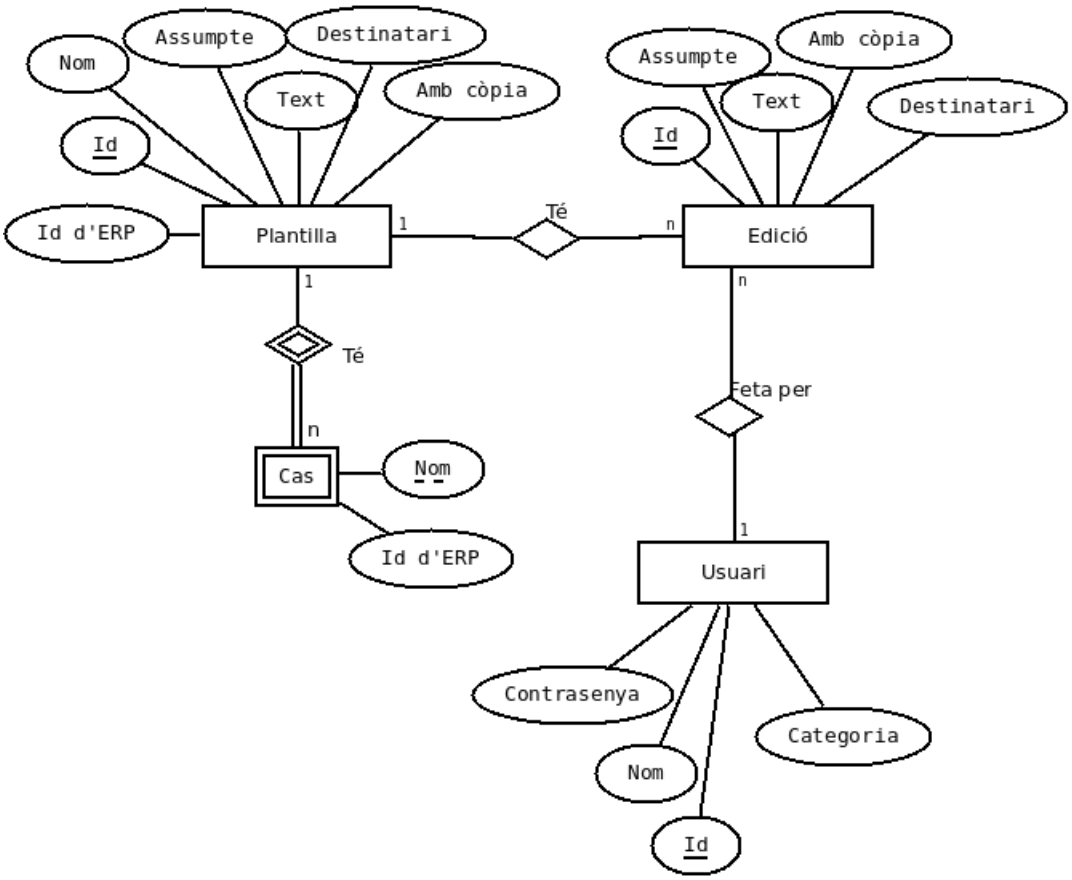

Figura 8.3: Model entitat-relació de les dades.

Com es pot veure al model, la complexitat del projecte no és causada per la quantitat d'informació que cal guardar. La complexitat del projecte és causada per la necessitat d'interacció entre els diferents elements del sistema i per la necessitat d'implementar una gestió de les edicions per tal que siguin fàcils i intuïtives de fer i alhora assegurar que els canvis siguin correctes i segurs.

Fitxes de casos d'ús

He escollit dos casos d'ús representatius de l'eina desenvolupada per fer-ne la fitxa de cas d'ús: l'edició d'una plantilla i pujar els canvis (edició) d'una plantilla a l'ERP:

CAS D'ÚS	Edició d'una plantilla
Versió	Visió anàlisi
Descripció	Un usuari vol editar una plantilla
Actors	Usuari bàsic, usuari avançat
Precondició	Usuari i plantilla donats d'alta al sistema
Flux principal	1. Buscar usuari 2. Comprovar usuari no deshabilitat 3. Buscar plantilla 4. Comprovar si la plantilla té edicions d'altres usuaris en curs 4.1.Si hi ha altres edicions: 4.1.1 Mostrar missatge d'avís d'altres edicions en curs 4.2 FSi 5. Si l'usuari és avançat: 5.1. Seleccionar tipus editor a utilitzar 6. Altrament: 6.1. Assignar editor a utilitzar l'editor HTML 7. FSi 8. Si l'usuari té una edició en curs: 8.1. Obtenir els valors de l'edició de l'usuari 9. Altrament: 9.1. Crear una nova edició. 10. Fsi 11. Assignar els valors de l'edició segons les dades introduïdes per l'usuari.
Postcondició	S'ha creat una nova edició de l'usuari per la plantilla seleccionada o bé s'ha actualitzat l'edició que hi havia en curs

CAS D'ÚS	Pujar canvis d'una plantilla a l'ERP
Versió	Visió anàlisi
Descripció	Un usari vol pujar canvis (nova versió) d'una plantilla a l'ERP
Actors	Usuari bàsic, usuari avançat
Precondició	L'usuari ha fet una edició d'una plantilla. Des que l'usuari va començar l'edició, no s'ha modificat la versió a productiu de la plantilla. Hi ha connexió amb algun ERP.
Flux principal	1. Buscar usuari 2. Comprovar usuari no deshabilitat 3. Buscar l'edició de l'usuari 4. Comprovar l'edició parteix de la versió vigent 5. Obtenir els casos de testeig de la plantilla 6. Mentre hi hagi casos de testeig i l'usuari no rebutgi el resultat: 6.1. Mostrar resultat de la renderització de la edició amb el cas actual 7. FMentre 8. Mostrar els ERPs pels quals es té connexió 9. Pujar els canvis a l'ERP seleccionat 10. Si l'ERP seleccionat és el de PRODUCCIÓ: 10.1. Eliminar l'edició de l'usuari 10.2. Afegir la nova versió al control de versions 10. FSi
Flux alternatiu	1. Si l'edició parteix d'una versió obsoleta llavors mostrar missatge d'avís, el contingut de la versió actual i el contingut de l'edició
Postcondició	S'ha pujat el contingut d'una edició a l'ERP seleccionat. Si l'ERP seleccionat és el de productiu, l'edició s'ha eliminat i la nova versió s'ha inclòs al control de versions.

8.2 Disseny

Un cop fet l'anàlisi del sistema, el disseny de la base de dades pel projecte és el següent:

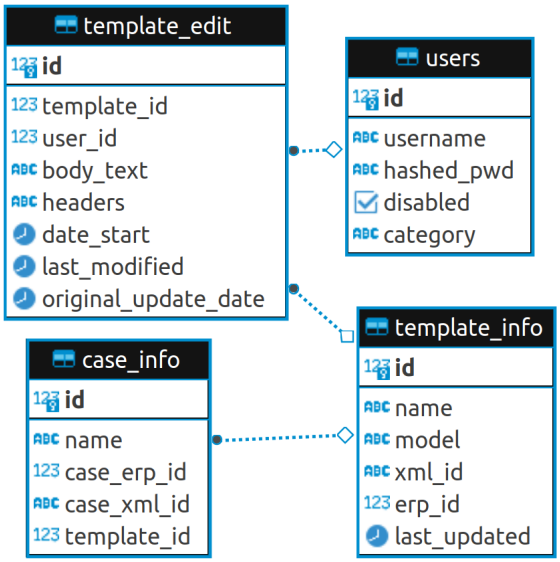

Figura 8.4: Taules de la base de dades.

Aquest disseny té una diferència respecte el model entitatrelació proposat, ja que no es persisteix a la Base de Dades totes les dades de la plantilla, sinó només certs camps (com el nom i l'ID a l'ERP) d'aquesta. Els altres camps de la plantilla es troben a la Base de Dades de l'ERP, i es consulten cada vegada per poder-ne tenir la versió més actualitzada. L'entitat plantilla del model es conforma de la informació continguda a la taula *template_info* juntament amb la que es consulta en cada moment a l'ERP.

Cal tenir en compte que l'eina dissenyada fa peticions a l'ERP per a obtenir la versió actual de les plantilles, i per tant també actua com a base de dades. Tot i que el model a l'ERP té molts més atributs, els que tenen rellevància per les funcionalitats a implementar són les següents:

També cal tenir en compte que el backend interacciona amb un repositori de git (un directori) per a guardar-hi les versions definitives de les plantilles i tenir el control de versions. Aquest repositori actua com una tercera base de dades on, per cada plantilla, s'emmagatzema un fitxer amb el cos de la plantilla i un altre fitxer amb les capçaleres.

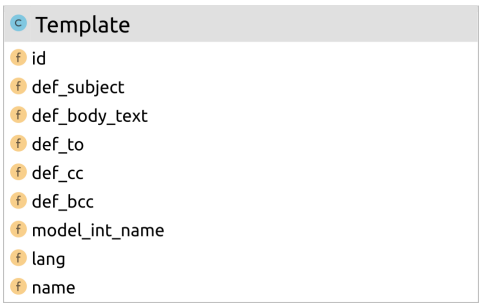

Figura 8.5: Esquema de la classe *Template* per a representar una plantilla llegida de l'ERP.

Utilitzant una representació semblant a la dels diagrames de seqüència, el següent esquema mostra la interacció dels diferents elements del sistema en el cas d'ús *Editar una plantilla*:

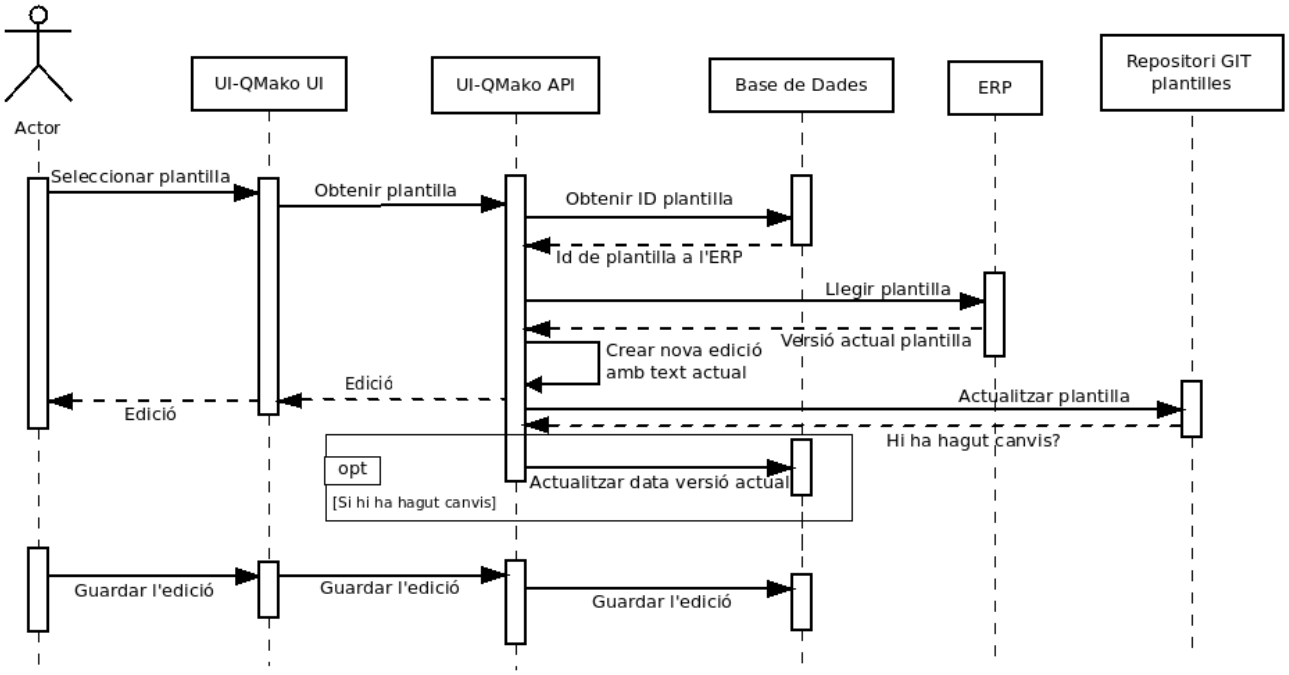

Figura 8.6: Representació de la interacció entre els diferents elements en editar una plantilla

Utilitzant també la mateixa representació, el següent esquema mostra la interacció dels diferents elements pel cas d'ús *Pujar canvis d'una plantilla a l'ERP*:

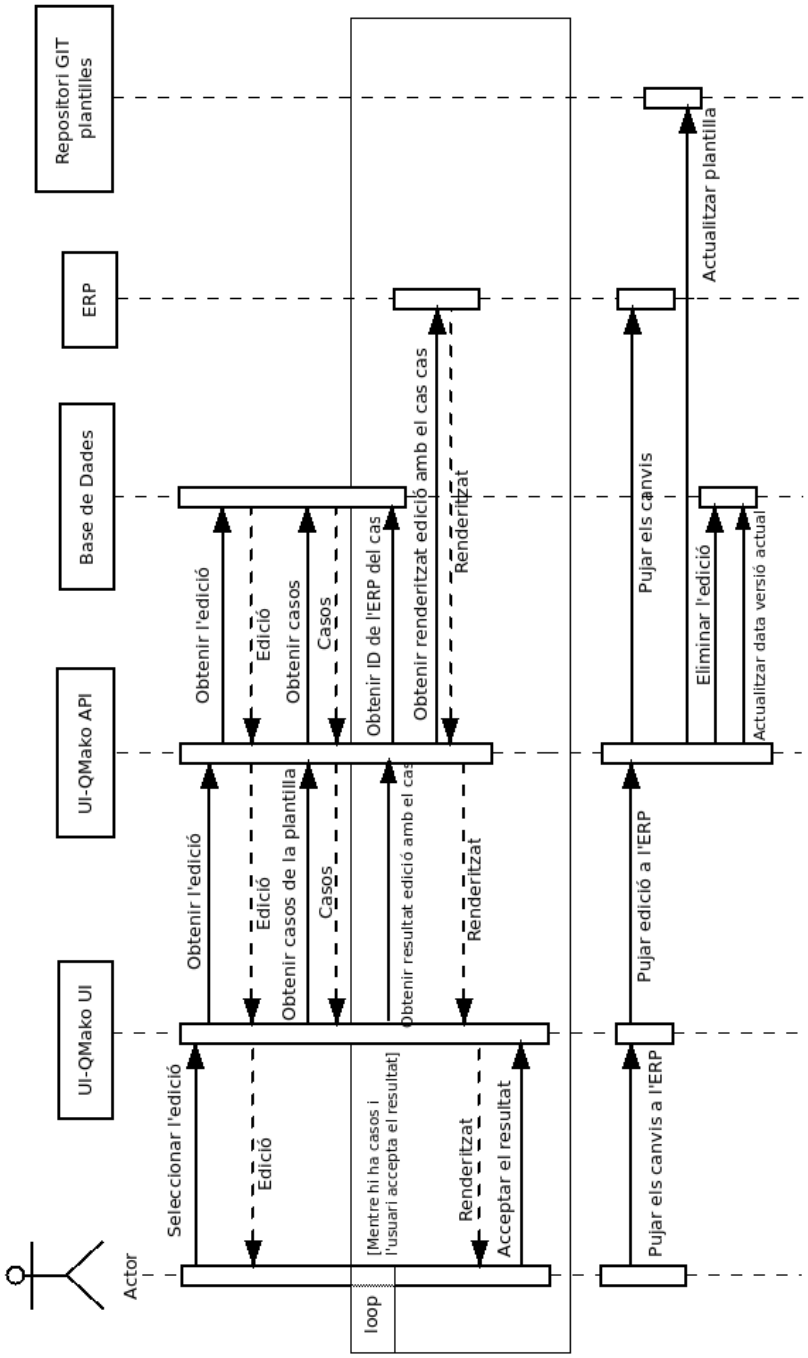

Figura 8.7: Representació de la interacció entre els diferents elements pujar canvis a l'ERP

Diagrama de classes

Per a la representació del digrama de classes he separat les classes que representen el model de dades de les classes que conformen l'app de l'API, ja que són dues parts ben diferenciades i on la única relació que hi ha és que l'API utilitza els models de dades. Per a facilitar-ne la lectura, no es mostren les funcions de les classes. Tampoc es mostren les classes pròpies del framework.

El diagrama de classes de les classes que conformen l'API és el següent:

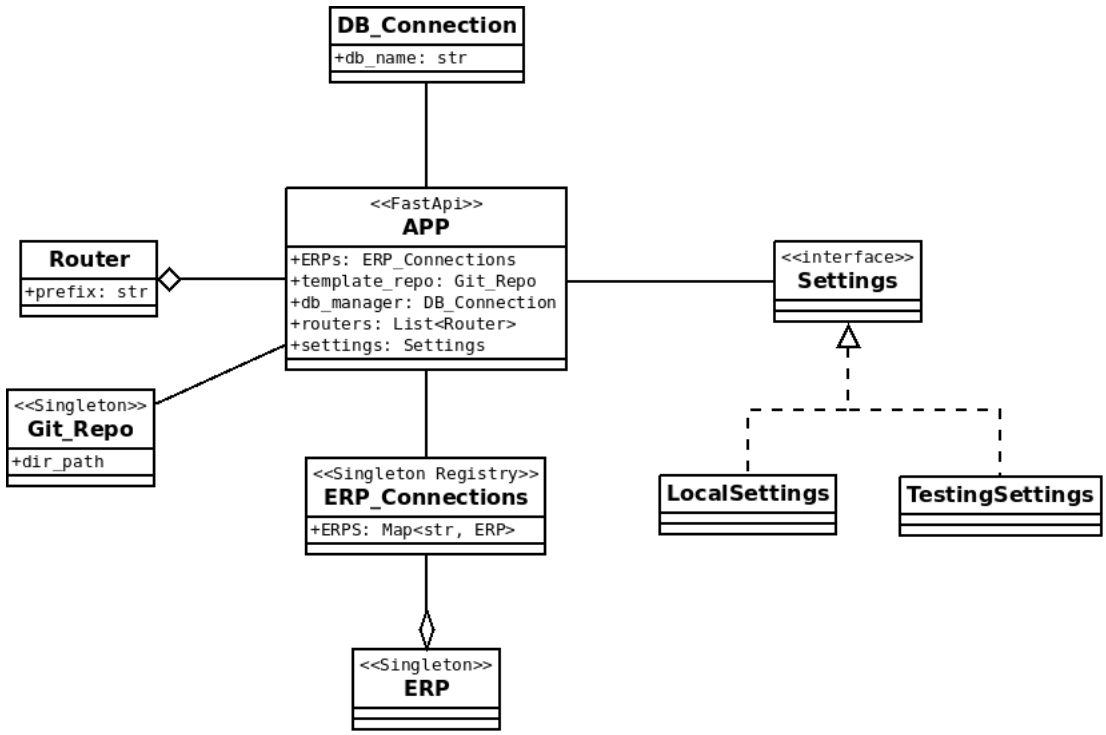

Figura 8.8: Diagrama de classes de l'API

I el diagrama de classes dels models de dades és el següent:

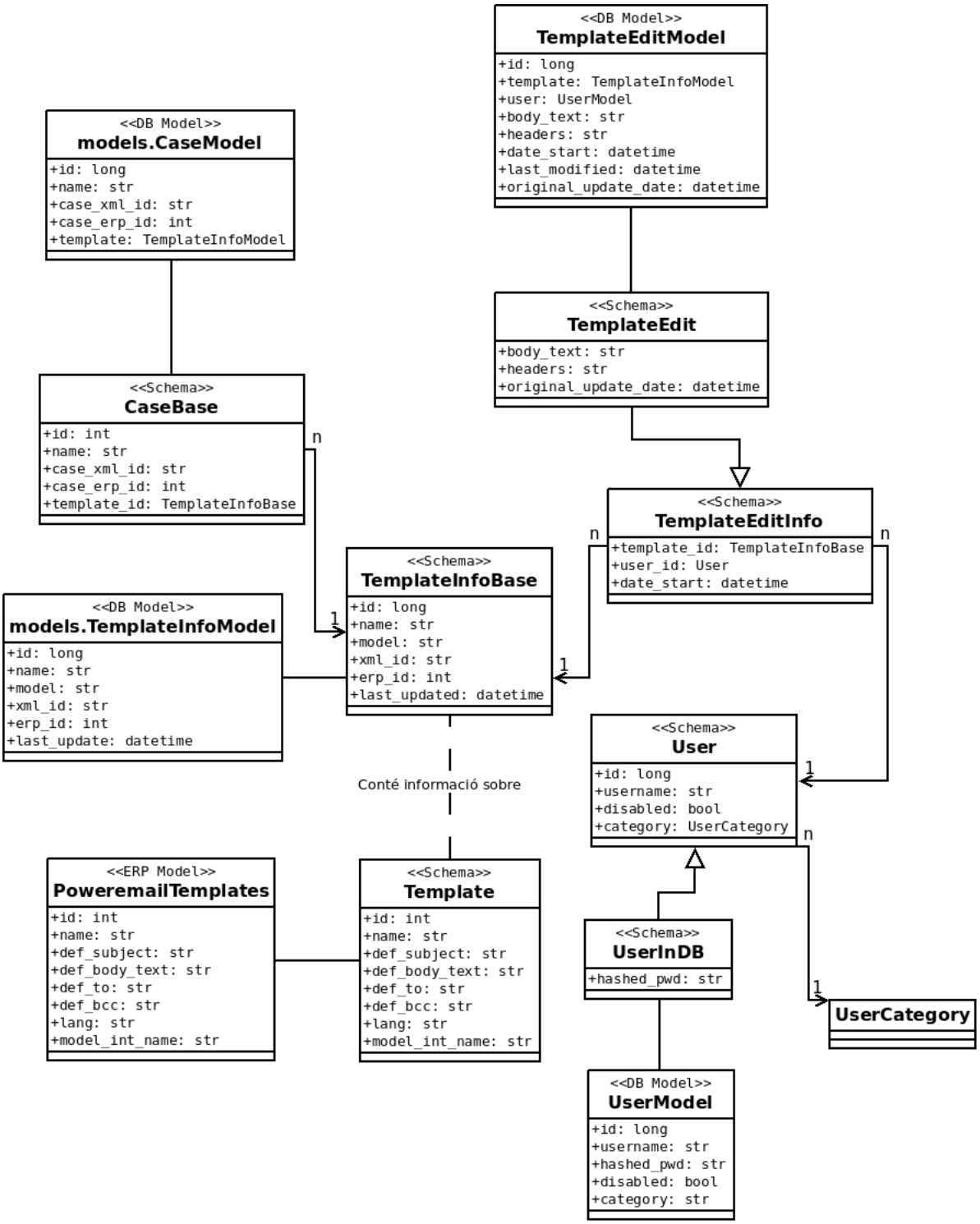

Figura 8.9: Diagrama de classes dels models de dades

Disseny del frontend

Degut a la sintaxi pròpia de React, tots els components implementats són funcions i del frontend implementat no se'n pot extreure un diagrama de classes.

Les diferents pantalles que caldrà implementar són les següents:

- Pàgina de log-in o registre.
- Pàgina principal amb el llistat de plantilles.
- Pàgina per afegir noves plantilles.
- Pàgina que mostri la versió actual d'una plantilla, amb els links per editar-la.
- Pàgina que permeti editar les capçaleres i el text d'una plantilla amb un editor de text pla.
- Pàgina que permeti editar les capçaleres i el text d'una plantilla amb un editor HTML WYSIWYG.
- Pàgina que permeti veure els casos d'una plantilla i afegir-ne de nous.
- Pàgina que permeti visualitzar el resultat de renderitzar una plantilla amb un cas.
- Pàgina que permeti iterar el resultat per cada cas d'una plantilla. Al final ha de permetre pujar els canvis a l'ERP.
- Pàgina de menú, que hi hagi links a fer log-out, afegir una nova plantilla i administrar permisos d'usuaris (si es té permís).

8.2.1 Patrons de disseny

Durant l'anàlisi del sistema he detectat diferents patrons de disseny aplicables. El primer patró clarament és el Model - Vista - Controlador, que permet desacoblar la vista de l'aplicació (la UI, en aquest cas) del model de dades (mòdul *models* del backend) del controlador (lògica del backend).

El segon patró utilitzat és el Singleton, utilitzat per a la classe de connexió amb el repositori GIT per a les plantilles. També utilitzo el patró Singleton per la connexió

de cada ERP al qual es connecta l'eina. L'ús d'aquest patró assegura que només hi ha una instància de la classe, necessari per evitar problemes de concurrència amb el repositori Git, per exemple, o per establir una sola connexió a l'ERP (múltiples connexions podrien arribar a saturar el servidor d'ERP).

Com que l'eina desenvolupada ha de poder establir connexió amb diferents ERP (per exemple l'ERP de productiu i el de testing), he utilitzat el patró *Singleton Registry*, també anomenat *Multiton* [Wikipedia 2021], que permet tenir un diccionari clau - instància Singleton. Gràcies a aquest patró, es pot implementar un Singleton per a cada ERP disponible.

Capítol 9

Implementació i proves

L'objectiu d'aquest apartat és exposar els problemes afrontats durant el desenvolupament de l'aplicació i la solució trobada, a part d'explicar com he preparat els entorns de desenvolupament.

9.1 Backend

La gestió dels paquets i mòduls utilitzats i les seves dependències en projectes de Python es gestionen dins d'entorns virtuals, de manera que cada projecte tingui el seu propi entorn, amb les llibreries necessàries instal·lades, i hi ha disponibles diferents eines per a gestionar aquests entorns. L'eina de gestió que he escollit per al desenvolupament del projecte ha estat *poetry* [python poetry 2021].

La instal·lació es pot fer executant la comanda:

```
$ pip install --user poetry
```

I per crear un nou projecte i inicialitzar l'entorn virtual:

```
$ poetry new nom-projecte
```

Amb aquesta comanda haurem creat una nova carpeta amb la següent estructura de fitxers:

On a la carpeta nom_projecte hi haurà el codi del projecte, i a la carpeta tests els tests d'aquest. El fitxer *pyproject.toml* és el que genera el *poetry* per a guardar totes les dependències del projecte. Un cop hem instal·lat algun paquet, es genera

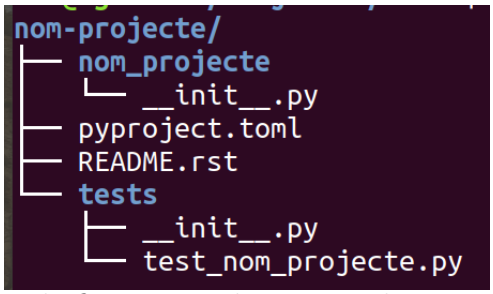

Figura 9.1: Estructura de fitxers creada en inicialitzar un projecte amb *poetry*.

un fitxer *poetry.lock* on queda constància de tots els paquets i mòduls instal·lats (inclosos les dependències de les nostres dependències) amb la seva versió.

Per executar comandes dins l'entorn virtual generat, cal que l'activem amb la comanda:

```
$ poetry shell
```

O bé precedir totes les comandes que vulguem executar amb la comanda *poetry run*:

```
$ poetry run python
```

El següent pas a fer és inicialitzar l'API utilitzant la classe de FastAPI i declarar el primer endpoint. Per tal d'utilitzar la metodologia TDD (test-driven-development) el següent pas és fer el *setup* dels tests.

Durant el desenvolupament, per tal de poder tenir diferents configuracions segons si l'execució és fa dins un entorn de desenvolupament o de test, he creat un mòdul de settings dins del projecte amb dues classes diferents de configuració:

- LocalSettings: utilitzat en entorn de desenvolupament: es carreguen les variables de configuració del fitxer *.env*. Aquest fitxer conté les contrasenyes, nom de la base de dades, clau secreta per la codificació de tokens, dades de connexió a l'ERP... i no s'inclou dins del control de versions.
- TestingSettings: utilitzat en l'entorn d'execució de tests. En aquest cas, les variables de configuració s'obtenen del fitxer *.env.test* i són variables de test.

En inicialitzar l'API, gràcies a una variable d'entorn, s'escull quina de les dues classes és la que s'utilitza per a la configuració.

A partir d'aquí ja es pot instal·lar l'ORM, declarar els models de dades i una funció de *setup* de la base de dades. Aquesta funció es crida durant el *setup* de l'API i s'encarregarà de crear els models a la base de dades (en cas que sigui necessari).

A mida d'anar afegint *endpoints* a l'API m'he adonat que no era una bona estratègia tenir-los tots al mateix fitxer. Aprofitant la classe APIRouter de FastAPI, que permet agrupar diferents rutes sota un mateix prefix, he decidit separar les operacions en tres routers diferents: els *endpoints* d'operacions amb usuaris, els de plantilles i els d'edicions.

FastAPI incorpora la validació de paràmetres d'entrada si s'utilitza notació de tipus en la declaració de les funcions de rutes. A més, també incorpora serialització i desserialització dels paràmetres d'entrada i sortida si s'implementen classes que heretin de la classe *BaseModel* de *pydantic*, classes a les quals s'anomena *Schemas*.

Així doncs, dins del projecte hi ha diferents mòduls:

- API: conté els fitxers que defineixen les diferents rutes de l'API.
- Models: conté els fitxers que defineixen els models de la base de dades del projecte, o bé els models que s'obtenen a través de consultes a l'ERP (ja que, en el fons, també és una base de dades pel que fa al projecte). També conté els mètodes per a fer les operacions CRUD sobre els models.
- Schemas: classes definides en Python per a la serialització de paràmetres d'entrada i sortida i també per a treballar amb els objectes dins del projecte. En llegir una instància de la base de dades es carrega en una classe de tipus *schema* per a poder fer-hi operacions o pel retorn.
- Útils: conté la lògica de les operacions.
- Errors: conté les excepcions del backend, tant les *HTTP exceptions* (respostes de l'API) com les excepcions internes.

I l'estructura de fitxers del projecte és la següent:

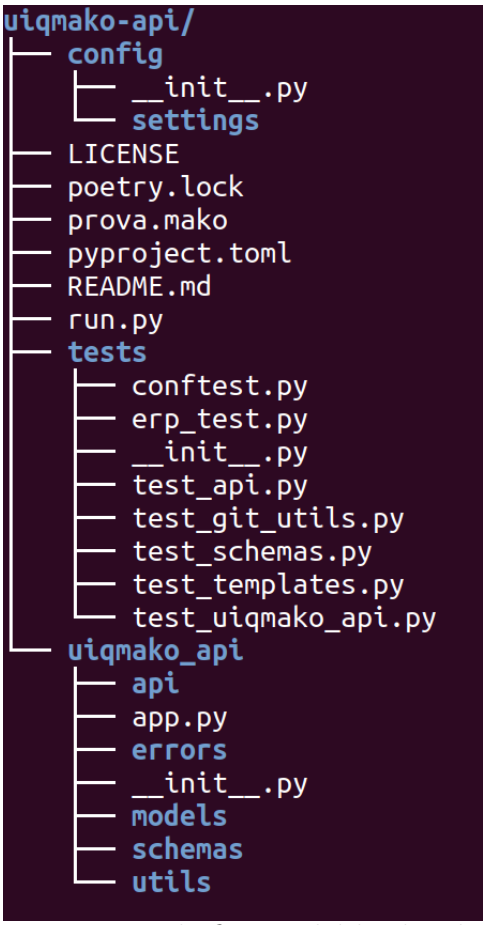

Figura 9.2: Estructura de fitxers del backend del projecte.

L'estructuració en aquests mòduls i l'aprofitament de les funcionalitats que proporciona FastAPI permet la implementació del patró Model-View-Controller, en els mòduls Models, API i útils, respectivament i a grans trets. Cal tenir en compte, que la View del backend són els *endpoints* de l'API, però la del projecte és el frontend.

Aquesta estructura permetria, fàcilment, canviar la base de dades que s'utilitza o afegir noves fonts de dades, per exemple, substituint o ampliant el mòdul de Models.

9.2 Frontend

Per preparar l'entorn del frontend he utilitzat l'eina *create react app*, instal·lable mitjançant el gestor de paquets de JavaScript npm [npm 2021] i la seva eina CLI *npx*. Per a inicialitzar un projecte de React amb aquesta eina només cal executar la comanda:

```
$ npx create-react-app my-app
```

Un cop executada, se'ns ha creat una carpeta anomenada *my-app* amb la següent estructura de fitxers:

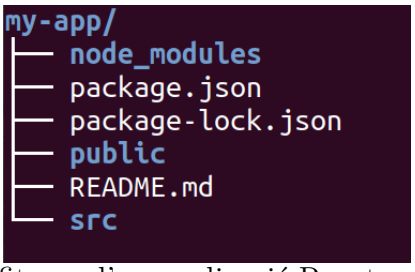

Figura 9.3: Estructura de fitxers d'una aplicació React creada amb *create-react-app*.

On a dins la carpeta *node_modules* hi ha totes les dependències del projecte i el fitxer *package-lock.json* conté el resum d'aquestes. El fitxer *package.json* conté la configuració de l'aplicació, el directori *public* conté les imatges, l'índex i el manifest de la pàgina web, i el directori *src* és el que contindrà els components React de l'aplicació.

Per executar l'aplicació en mode desenvolupament només cal executar la comanda següent dins del directori *my-app*:

```
$ npm start
```

Un cop acabat el desenvolupament, per generar els fitxers que conformaran la pàgina web, només cal executar:

```
$ npm run build
```

I haurà aparegut una carpeta *build* amb tots els fitxers HTML, CSS i JavaScript optimitzats que caldrà posar al servidor de producció per poder fer funcionar l'aplicació, així com els fitxers que contenen els arxius multimèdia.

9.3 Reptes

Un primer repte que he hagut de solucionar durant el desenvolupament del projecte és permetre l'edició de les plantilles amb un editor HTML WYSIWYG. No es pot mostrar directament el text de la plantilla a l'editor perquè aquest envolta cada element del text d'etiquetes HTML i elimina els salts de línia. Aquestes dues manipulacions fan que el codi Python de les plantilles deixi de ser interpretable.

La solució que he trobat és dividir el text de les plantilles entre parts de Python i parts d'HTML. Per a fer-ho, al backend utilitzo una expressió regular que divideix el text en parts, i la crida a l'API retorna el text dividit, Al frontend utilitzo l'editor WYSIWYG per mostrar les parts HTML i un camp de text simple per a les parts de codi Python de la llista rebuda.

```
def parse_body_by_language(full_text):
    """
    Function to split mako template text by language
    :param full_text: mako template text
    :return: a list of tuples, where the first element is 'python' or 'html'
    and the second element is a fragment of full_text written in said language
    """
    python_reg = "((<%)[\s\S]*?(%>))|((^[\s]*%[^>][\S ]*))"
    rex = re.compile(python_reg, re.MULTILINE)
    parts = []
    current_pos = 0
    for match in rex.finditer(full_text):
        start = match.start()
        if start != current_pos:
            html_text = full_text[current_pos:match.start()].strip()
            if html_text:
                parts.append(('html', html_text.replace('\n','')))
        parts.append(('python', match.group()))
```

```
        current_pos = match.end()
    if current_pos != len(full_text) or not parts:
        html_text = full_text[current_pos:].strip()
        if html_text:
            parts.append(('html', html_text))
    return parts
```

Listing 8: Funció del backend que divideix el cos de la plantilla entre parts Python i parts HTML.

En la interfície d'usuari, les diferents parts es mostren en un camp de text o a l'editor segons correspongui:

Figura 9.4: Interfície d'usuari mostrant el codi Python en camps de text i el codi HTML a l'editor HTML

Un cop fetes les edicions, s'envien també separades per tipus de llenguatge a l'API, que s'encarrega de tornar a composar el text. Per a fer-ho, he declarat un *schema*

anomenat *RawEdit*, que en inicialitzar-se processa la llista de les diferents parts per a construir el text sencer. A més, per tal de fer més comprensible la part HTML, utilitzo la funció *prettify* de la llibreria *Beautifulsoup*, que fa una identació de l'HTML.

```
class RawEdit(BaseModel):
    by_type: str = None
    def_body_text: str = None
    headers: str = None

    def __init__(self, by_type, def_body_text, headers):
        super(RawEdit, self).__init__()

        self.by_type = json.loads(by_type)
        self.headers = headers
        self.def_body_text = def_body_text
        self.compose_text_types()

    def compose_text_types(self):
        if self.by_type:
            full_text = ''
            for type, text in self.by_type:
                if type == 'html':
                    text = BeautifulSoup(
                        text, "html.parser"
                    ).prettify(formatter=None)
                full_text += text + '\n'
            self.def_body_text = full_text
```

Listing 9: Classe RawEdit, que encapsula els paràmetres d'una edició que rep l'API, i composa el text complert si rep una edició feta per parts.

A part de retornar el text separat pel llenguatge, l'API també retorna el text sencer de la plantilla, i és el que s'edita quan s'utilitza l'editor de text pla de la UI.

Un segon repte que he hagut de solucionar és la gestió de la caducitat del *token* d'autenticació amb l'API. En la interfície d'usuari s'emmagatzema el *token* que cal

enviar en cada petició, però aquests tenen una caducitat. En el frontend, per no haver de comprovar en cada petició si la resposta de l'API informa que el token està caducat, he utilitzat la funcionalitat de React *Context* [Facebook 2021a] per emmagatzemar el token i l'hora i dia de l'obtenció. Llavors, sabent el temps de caducitat, quan l'usuari fa servir la interfície d'usuari, comprovo que no hagi caducat. També he utilitzat el *Context* de React per emmagatzemar els permisos de l'usuari autenticat i poder restringir l'accés a certs menús o elements segons convingui.

Un altre repte que he hagut de solucionar és el renderitzat de les plantilles amb casos (instàncies dels models) de l'ERP, ja que l'accés a les variables que es volen mostrar, en alguns casos, només són accessibles des de dins del codi d'aquest software. Com que moltes de les plantilles necessiten accedir a models interns de l'ERP, no és possible renderitzar-les simplement utilitzant les funcionalitats de la llibreria Mako. El repositori Oomakotest (predecessor de l'eina desenvolupada durant aquest projecte) soluciona aquest impediment utilitzant una execució en local de l'ERP, i fent les operacions dins d'aquesta execució. Aquesta solució no és viable per dos motius: l'ERP funciona en Python 2 (i és incompatible amb Python 3) i perquè un dels objectius principals del projecte és evitar el requeriment de tenir un ERP funcionant (connectat a una Base de Dades completa).

La solució a la qual he arribat és desenvolupar un mòdul del propi ERP amb una funció que rep un text mako, el nom d'un model i la id de la instància d'aquest model pel qual es vol renderitzar la plantilla, i retorna el text ja renderitzat. Amb aquesta solució, a més, asseguro que el resultat del renderitzat és el mateix que en l'entorn de producció, i evito diferències que es podrien produir per l'ús de versions diferents de les llibreries.

```
# -*- coding: utf-8 -*-

from osv import osv, fields
from tools import config
from mako.template import Template

class UiqmakoHelper(osv.osv_memory):
    _name = 'som.uiqmako.helper'

    def render_mako_text(self, cursor, uid, id, message, model,
                         recid, context={}):
```

```
        """
        :param cursor: db cursor
        :param uid: active user
        :param id: instance id
        :param message: mako text
        :param model: source model for the mako template
        :param recid: model's instance id
        :param context: context
        :return: html result from rendering the message with the model recid
        """
        context.update({'browse_reference': True})
        object = self.pool.get(
            model
        ).browse(cursor, uid, recid, context)
        env = context.copy()
        env.update({
            'user': self.pool.get('res.users').browse(cursor, uid, uid,
                                                      context),
            'db': cursor.dbname
        })

        templ = Template(message, input_encoding='utf-8')
        extra_render_values = env.get('extra_render_values', {}) or {}
        values = {
            'object': object,
            'peobject': object,
            'env': env,
            'format_exceptions': True,
        }
        values.update(extra_render_values)
        result = templ.render_unicode(**values)
        return result

UiqmakoHelper()
```

Listing 10: Classe dins l'ERP que implementa el renderitzat d'una plantilla.

Aquest últim fragment de codi, que també he desenvolupat jo, està inclòs en el repositori de l'ERP, i no en els repositoris del projecte.

9.4 Proves

Proves del Backend

Les proves del backend les he implementat en tests unitaris i d'integració, usant una classe *mock* per l'ERP, que sense connectar-se a l'ERP retorna valors controlats i objectes de testeig.

Els tests es poden executar amb la comanda següent:

```
$ poetry run pytest --cov=uiqmako_api --cov-report html
```

A part de poder veure el resultat, també extreu un informe de cobertura del codi en format HTML. La cobertura aconseguida, per fitxer, es mostra a la Figura 9.5.

Tal com es pot veure a l'informe generat per *pytest*, la cobertura total és del 81%. Els fitxers menys testejats són els que conten funcions de connexió amb l'ERP (*utils/erp.py* i *models.erp_models.py*) o que fan crides a l'ERP (*api/edits.py* i *utils/edits.py*). Això és degut a la dificultat de poder crear tests fiables i robusts mockejant les crides a aquest software extern.

Proves del Frontend

Tal com he comentat a l'apartat de metodologia, les proves del frontend les he fet manualment interaccionant amb la interfície i comprovant que el resultat era l'esperat. També he comprovat que en cas d'error, es mostrés un missatge explicatiu a l'usuari.

Com a treball futur seria interessant poder afegir tests automatitzats a la UI, utilitzant llibreries que permeten fer-ho, com pot ser *Cypress*.

Coverage report: 81%

Module ↓	statements	missing	excluded	coverage
uiqmako_api/utils/users.py	37	3	0	92%
uiqmako_api/utils/templates.py	49	6	0	88%
uiqmako_api/utils/git.py	36	0	0	100%
uiqmako_api/utils/erp.py	92	48	0	48%
uiqmako_api/utils/edits.py	41	21	0	49%
uiqmako_api/utils/__init__.py	0	0	0	100%
uiqmako_api/schemas/users.py	40	1	0	98%
uiqmako_api/schemas/templates.py	64	2	0	97%
uiqmako_api/schemas/edits.py	34	0	0	100%
uiqmako_api/schemas/__init__.py	0	0	0	100%
uiqmako_api/models/users.py	33	0	0	100%
uiqmako_api/models/templates.py	83	8	0	90%
uiqmako_api/models/erp_models.py	19	6	0	68%
uiqmako_api/models/edits.py	50	2	0	96%
uiqmako_api/models/__init__.py	27	0	0	100%
uiqmako_api/errors/http_exceptions.py	22	6	0	73%
uiqmako_api/errors/exceptions.py	21	4	0	81%
uiqmako_api/errors/__init__.py	0	0	0	100%
uiqmako_api/app.py	26	1	0	96%
uiqmako_api/api/users.py	23	2	0	91%
uiqmako_api/api/templates.py	39	13	0	67%
uiqmako_api/api/edits.py	46	26	0	43%
uiqmako_api/api/dependencies.py	36	10	0	72%
uiqmako_api/api/api.py	35	5	0	86%
uiqmako_api/api/__init__.py	2	0	0	100%
uiqmako_api/__init__.py	1	0	0	100%
Total	**856**	**164**	**0**	**81%**

Figura 9.5: Cobertura en tests dels fitxers.

Capítol 10

Implantació i resultats

Autenticació i Registre

En accedir a l'adreça de l'aplicació apareix la pantalla d'autenticació de l'usuari:

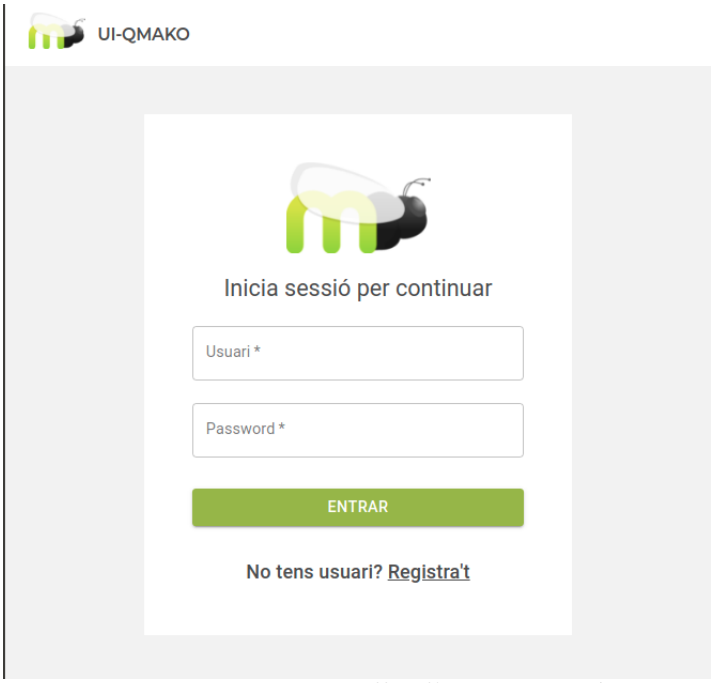

Figura 10.1: Pantalla d'autenticació

Des d'aquesta pantalla podem accedir a registrar-nos a l'aplicació, i s'assignaran els permisos bàsics al nou usuari. En la pantalla de registre es fan diferents validacions: que la contrasenya tingui un mínim de 8 caràcters, que es repeteixi correctament la contrasenya per validar-la i que el nom d'usuari no estigui ja registrat.

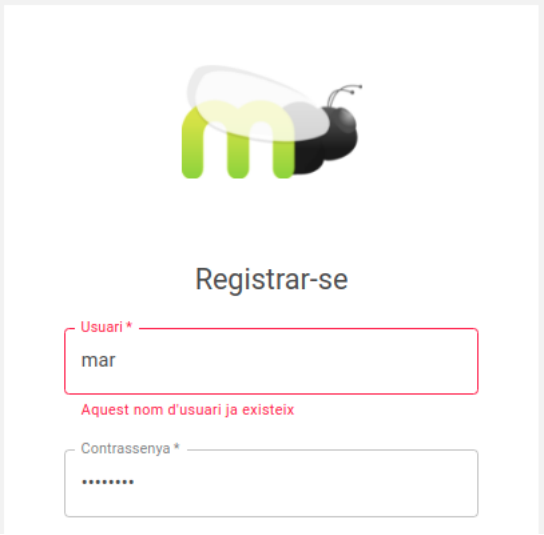

Figura 10.2: Nom d'usuari ja existeix.

Figura 10.3: Errors en el registre: contrasenya no vàlida i contrasenyes no iguals.

Un cop l'usuari està registrat podem veure a la base de dades que està creat, amb permisos bàsics i sense estar deshabilitat:

	id [PK] serial	username character varying(255)	hashed_pwd character varying(255)	disabled boolean	category character varying(255)
1	1	mar	$2b$12$7QL/XLvpOYvu	FALSE	admin
2	2	admin	$2b$12$GLqiG8Y2Pwca	FALSE	admin
3	39	mar_jene	$2b$12$sznN9BHgOB5.	FALSE	basic_user

Figura 10.4: Usuari *mar_jene* creat amb permisos bàsics.

I les crides que s'han realitzat a l'API són la de crear l'usuari i la d'obtenir la llista de plantilles (pantalla principal).

```
INFO:     127.0.0.1:56582 - "POST /users HTTP/1.1" 200 OK
INFO:     127.0.0.1:56582 - "OPTIONS /templates HTTP/1.1" 200 OK
INFO:     127.0.0.1:56582 - "GET /templates HTTP/1.1" 200 OK
INFO:     127.0.0.1:56582 - "OPTIONS /templates HTTP/1.1" 200 OK
```

Figura 10.5: Crides a l'API quan un usuari es crea.

Afegir una plantilla

La pantalla principal de l'aplicació, que apareix en fer LogIn o registrar-se correctament mostra la llista de les plantilles disponibles, ordenades creixentment per nom:

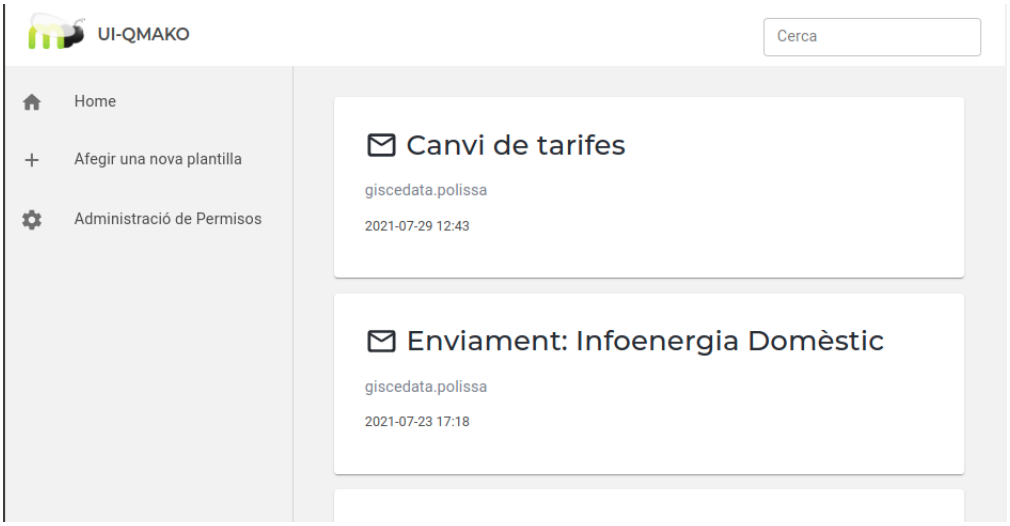

Figura 10.6: Pantalla principal de l'aplicació.

Per cada plantilla es mostra el nom, el model sobre el que es renderitza i la última data de consulta.

Des de la vista principal es pot filtrar, utilitzant el camp de cerca, per una cadena de text que aparegui al nom o al model de la plantilla:

Figura 10.7: Filtrat en la cerca.

Per afegir una plantilla hi ha la opció al menú lateral, i es mostra una vista que permet, a partir d'un XML ID, afegir una nova plantilla, tal com es mostra a la Figura 10.8. Si la plantilla s'ha afegit correctament, apareix el resum d'aquesta, tal com es mostra a la Figura 10.9. En cas que la plantilla ja existeixi a l'aplicació, es mostra un missatge indicant-ho, tal com es pot veure a la Figura 10.10.

Figura 10.8: Formulari per afegir una nova plantilla.

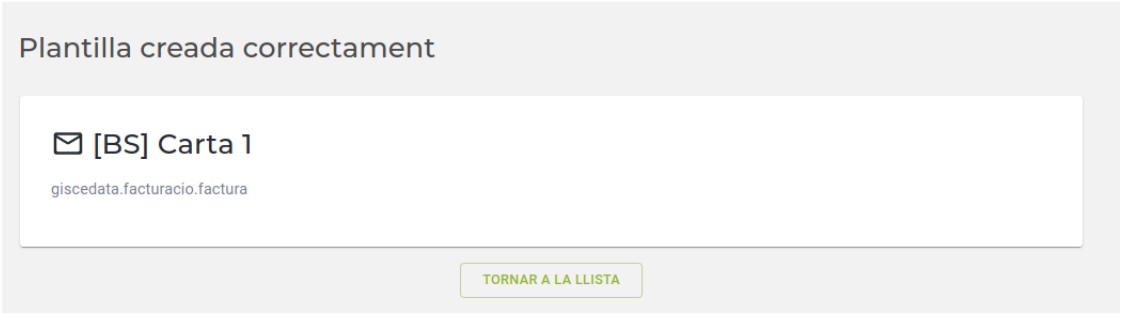

Figura 10.9: Plantilla afegida correctament.

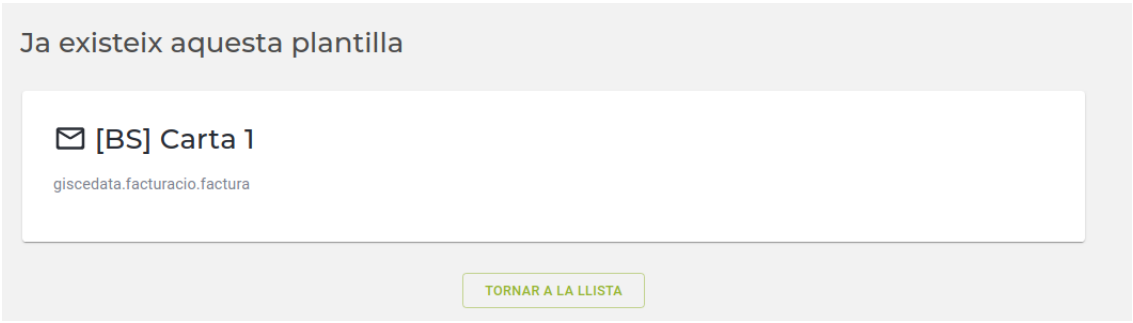

Figura 10.10: La plantilla afegida ja estava registrada.

Des de la pantalla principal, en clicar en una de les plantilles s'obre un *modal* amb la informació més detallada de la plantilla. Aquesta informació es consulta a l'ERP, i si l'aplicació detecta que és una nova plantilla o bé hi ha hagut canvis des de l'última consulta (canvis fets des de l'ERP), n'actualitza la versió al repositori de plantilles.

Edició de plantilles

Des d'aquesta vista, clicant el botó del llapis, podem editar la plantilla amb l'editor que triem, tal com es pot veure a la Figura 10.11. Quan l'usuari autenticat no té permisos d'edició Python només es mostra el botó de l'editor HTML.

En aquesta vista podem veure el nom, assumpte, destinataris, com es selecciona l'idioma de la plantilla i a quin model aplica. També es mostra una previsualització del text actual.

A la vista de l'editor HTML es mostra el text dividit entre parts de codi Python i parts de codi HTML, tal i com es pot veure a la Figura 10.13. Les parts de codi Python es mostren amb l'edició deshabilitades per usuaris amb permisos bàsics, tal com es mostra a la Figura 10.14.

Des de l'editor HTML també es pot editar el codi HTML, clicant a la icona amb el símbol </> de la barra d'eines, tal com es pot veure a la Figura 10.15. En cas d'haver-hi edicions d'altres usuaris en curs, es mostra una alerta, tal com es pot veure a la Figura 10.12.

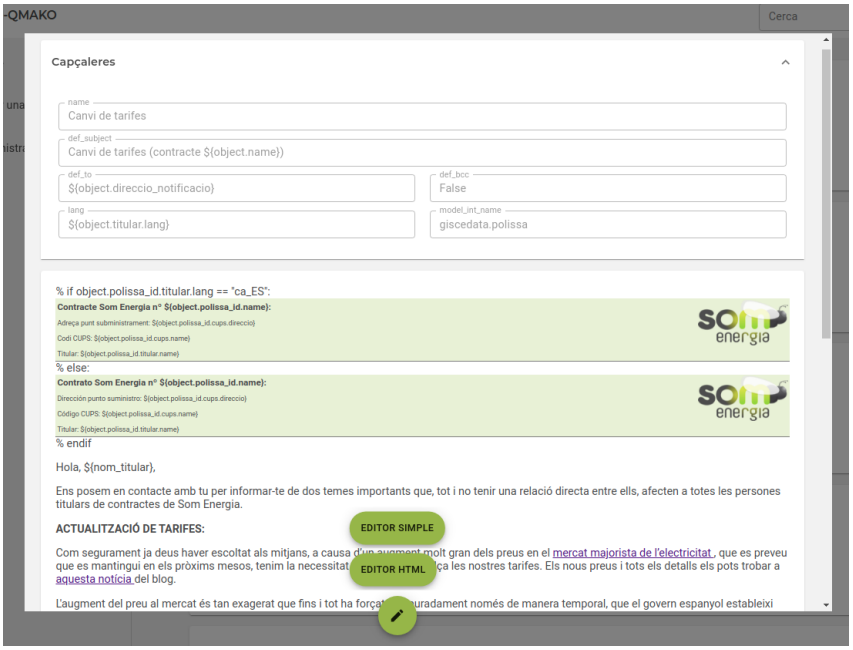

Figura 10.11: Vista d'una plantilla.

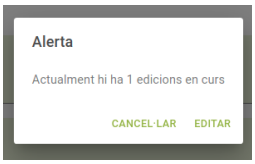

Figura 10.12: Missatge d'alerta d'altres edicions en curs.

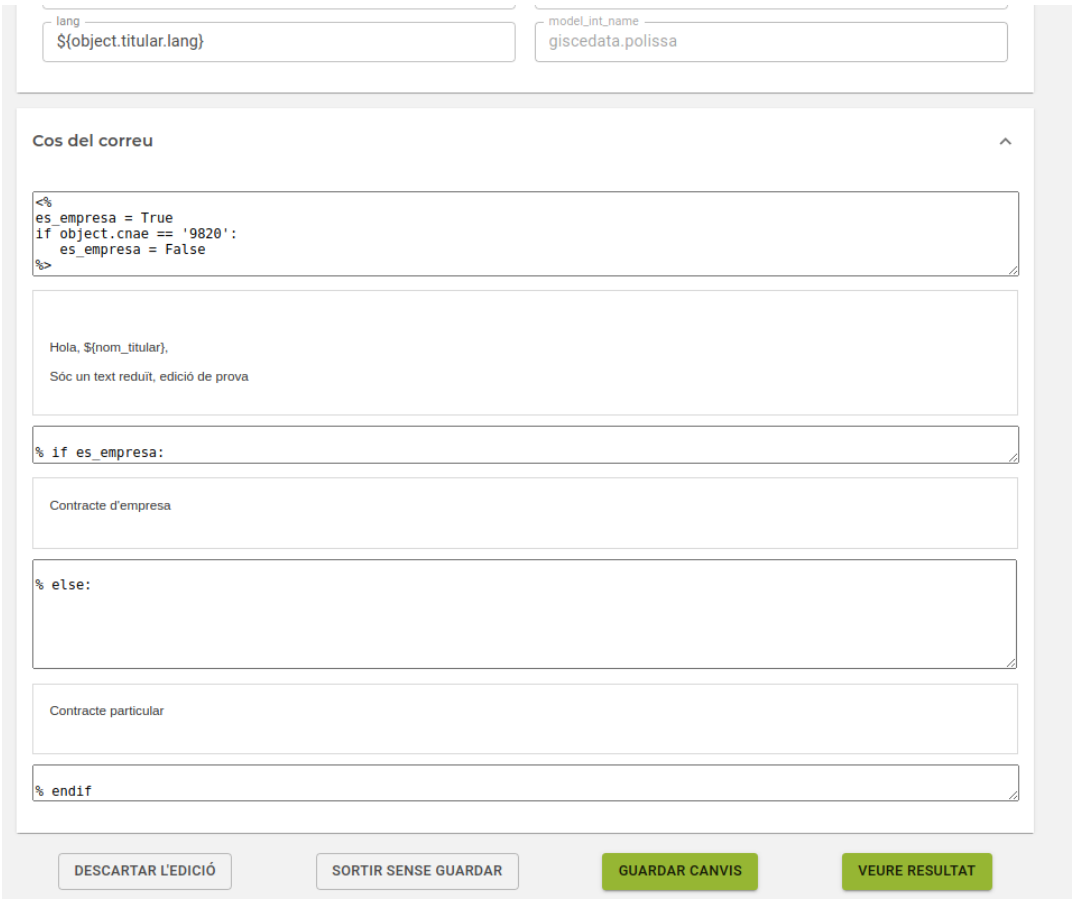

Figura 10.13: Vista de l'editor HTML amb el codi Python habilitat per a l'edició.

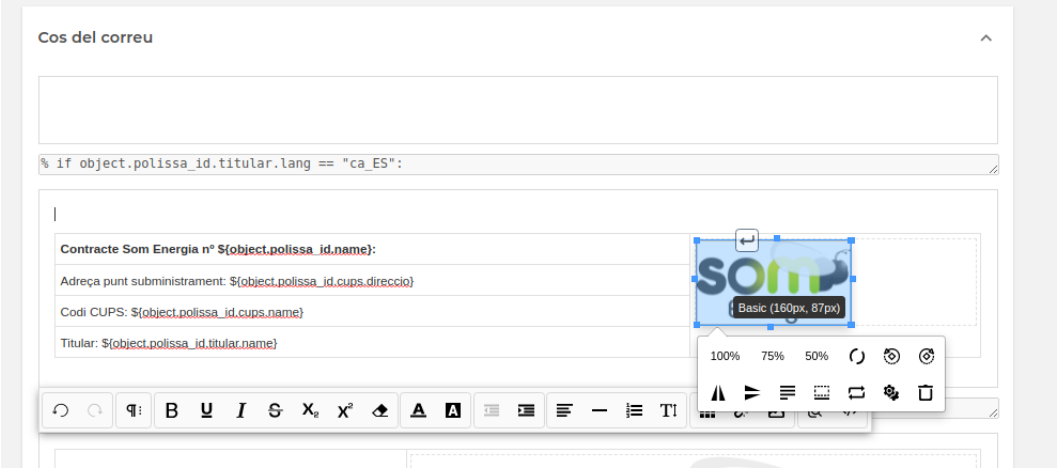

Figura 10.14: Vista de l'editor HTML amb el codi Python deshabilitat per a l'edició.

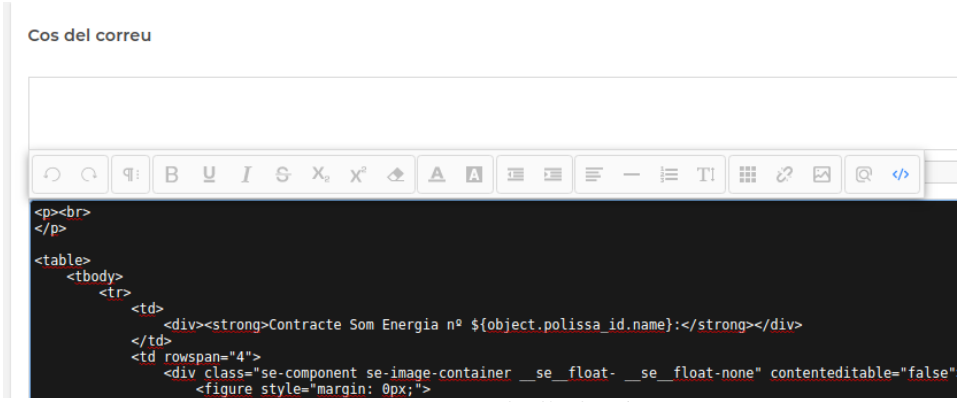

Figura 10.15: Vista de l'edició HTML.

La vista de l'editor simple (editor de text pla) es pot veure a la figura 10.16.

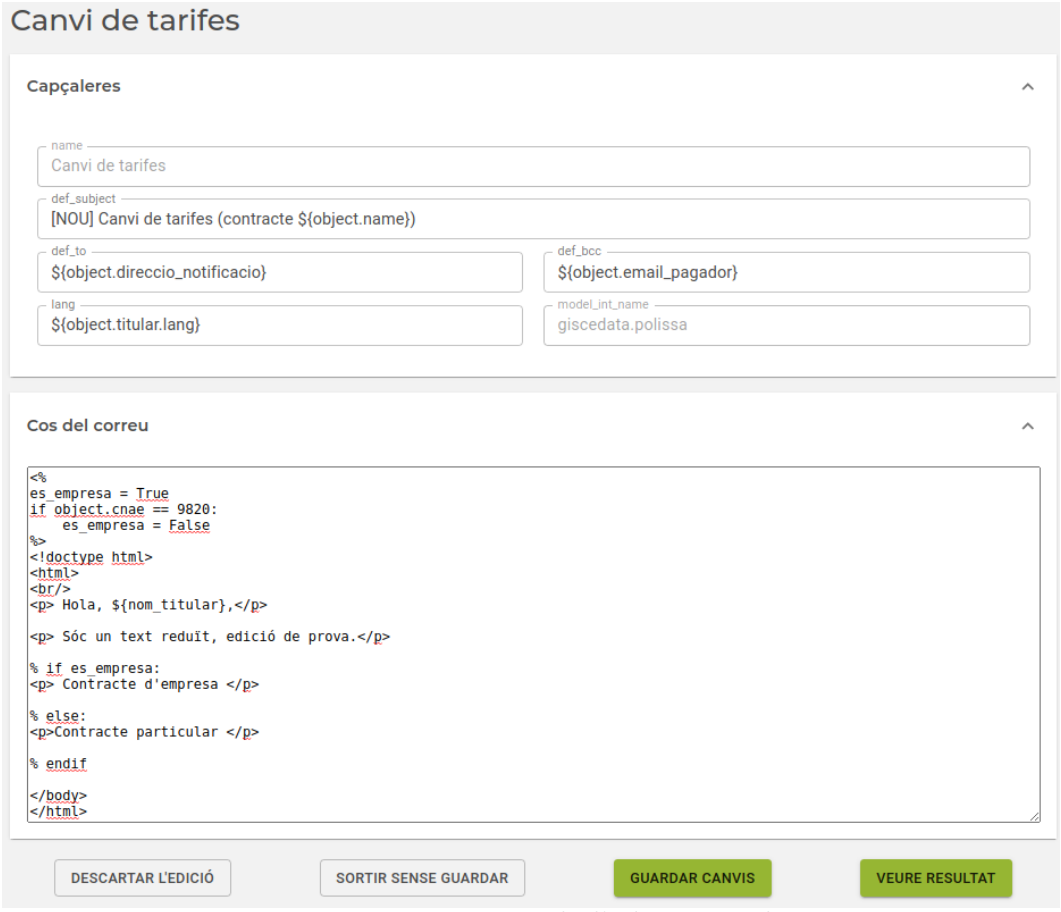

Figura 10.16: Vista de l'editor simple.

Veure el resultat i pujar els canvis

Un cop realitzats els canvis, es poden guardar per a continuar en un altre moment, o també es pot veure el resultat de renderitzar l'edició amb un cas de l'ERP. En el mateix diàleg que mostra la llista dels casos disponibles se'n poden afegir de nous, i només si n'hi ha com a mínim un es pot revisar tots els casos i pujar els canvis a l'ERP.

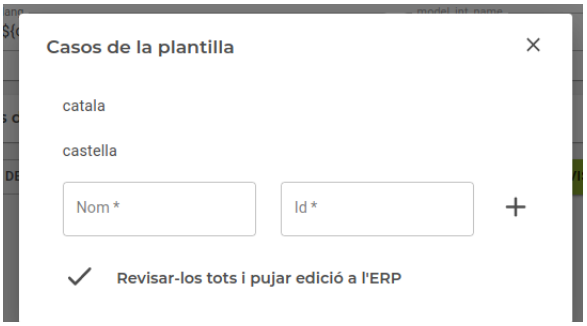

Figura 10.17: Casos disponibles per a la plantilla.

Un cop revisats tots els resultats, apareix un botó de pujar els canvis a l'ERP. Aquest botó obre un diàleg que deixa escollir en quin ERP dels quals té connexió l'API es volen pujar els canvis. En cas d'escollir l'ERP de producció s'elimina l'edició de l'usuari i automàticament s'actualitza el repositori de les plantilles amb la nova versió.

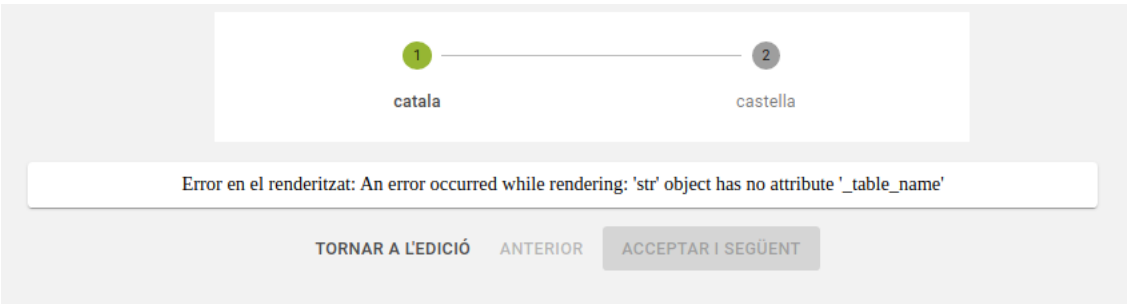

Figura 10.18: Error en el renderitzat de l'edició amb el cas seleccionat.

Figura 10.19: Resultat del renderitzat de l'edició amb un cas de l'ERP.

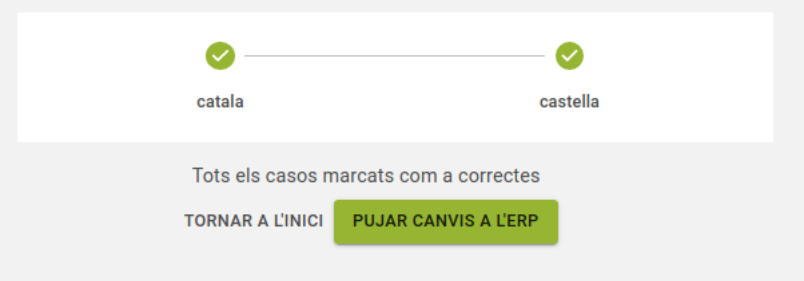

Figura 10.20: Final de la iteració, amb tots els casos marcats com a correctes.

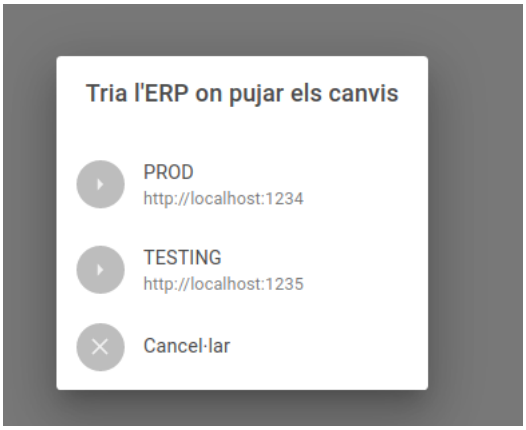

Figura 10.21: Diàleg per seleccionar l'ERP.

```
Mar Jene o [master] update headers for Canvi de tarifes
Mar Jene o update body_text for Canvi de tarifes
```

Figura 10.22: Commits al repositori de les plantilles per actualitzar-ne la versió

```
@@ -1,4 +1,4 @@
 cc: False
 to: ${object.direccio_notificacio}
-subject: Canvi de tarifes (contracte ${object.name})
-bcc: False
+subject: [NOU] Canvi de tarifes (contracte ${object.name})
+bcc:
```

Figura 10.23: Diferències del commit per actualitzar les capçaleres de la plantilla.

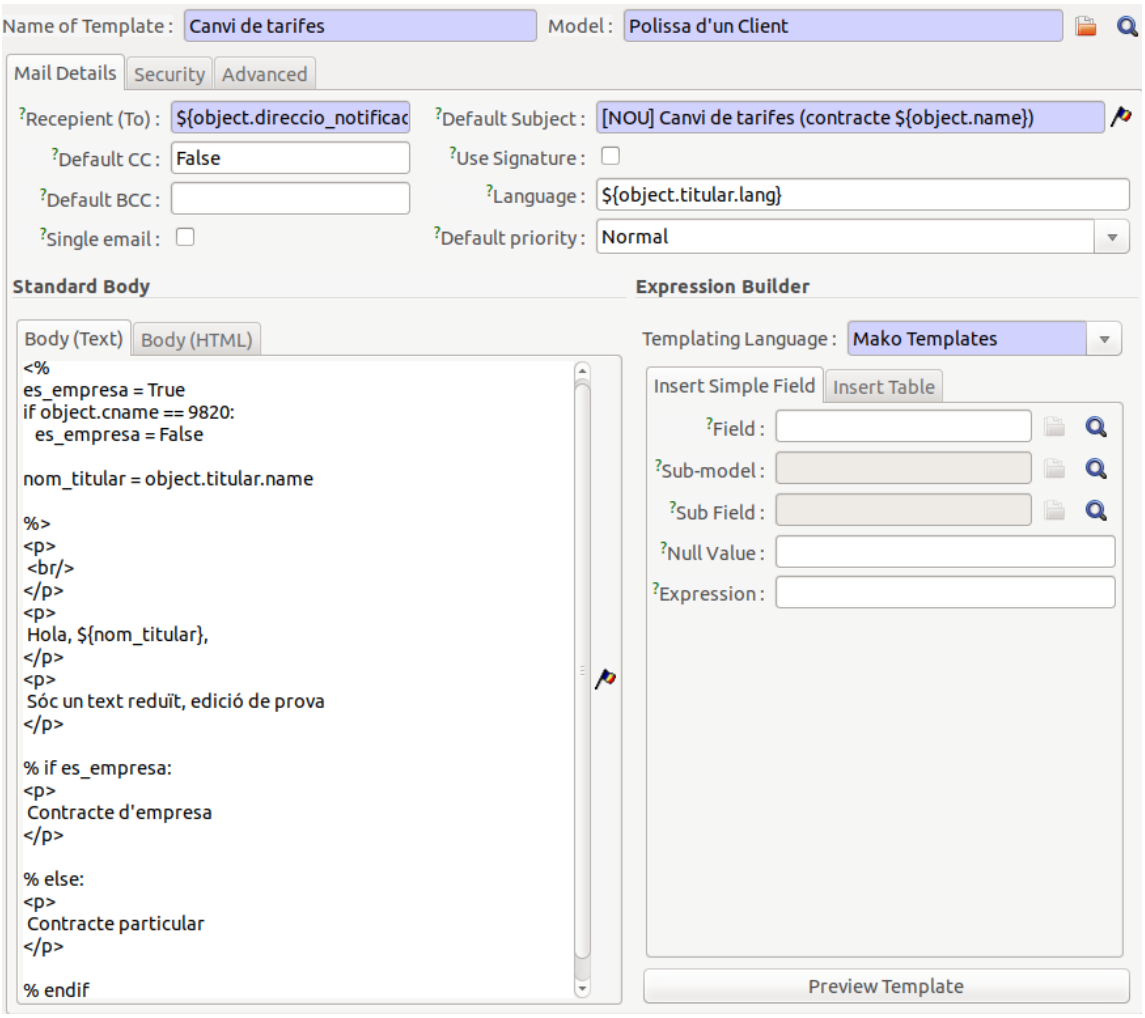

Figura 10.24: Nova versió de la plantilla a l'ERP.

Administració d'usuaris

Els usuaris amb permisos d'administració tenen habilitada la opció del menú d'Administració de permisos. Quan s'hi accedeix es mostra un llistat de tots els usuaris, amb els permisos que tenen actualment i si estan deshabilitats o no. Utilitzant el camp de cerca de la part superior esquerra es poden filtrar per nom d'usuari. En aquest llistat no apareix l'usuari *admin*, que es crea per defecte quan s'instal·la l'API, per evitar que se li treguin els permisos d'administració. Aquesta vista es pot veure a la Figura 10.25. A part de comprovar els permisos dels usuaris des del frontend, les crides a l'API també estan protegides: si el token que arriba amb la petició correspon a un usuari sense permisos d'autenticació l'API retorna el codi d'error 401 amb un missatge indicant que l'usuari no té permisos.

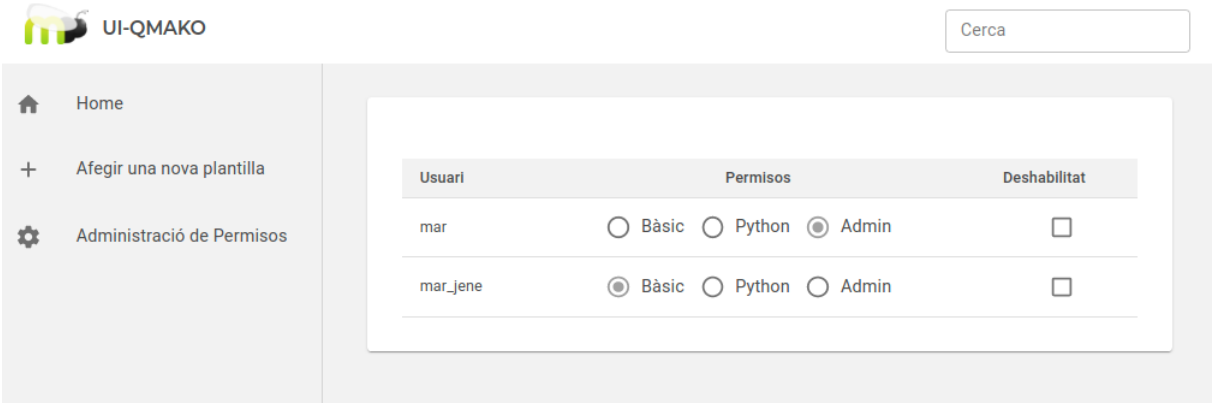

Figura 10.25: Vista d'administració de permisos.

Tot el codi desenvolupat es pot consultar a Github. L'adreça del repositori del backend és *https://github.com/Som-Energia/uiqmako-api* i la del frontend *https://github.com/Som-Energia/uiqmako-ui*. En cada repositori s'ha congelat la versió entregada a la branca anomenada *v0*.

Capítol 11

Conclusions

Aquest apartat té com a finalitat valorar el grau d'assoliment dels objectius i el coneixement adquirit durant el procés.

Un cop finalitzat el projecte, l'eina UI-QMako compleix els objectius marcats:

- Permet editar plantilles Mako desacoblant-ne les diferents parts: text, estil i codi Python.
- Permet testejar els canvis (edicions) amb casos reals de l'ERP.
- Incorpora un control de versions per a les versions definitives.
- Permet pujar les edicions a l'ERP.

Pel que fa als requisits funcionals, s'han implementat tots els classificats com a necessaris, i bona part dels classificats com a desitjables. Queden pendents els requeriments d'eliminar plantilles i casos de l'eina, i poder comparar el renderitzat de l'edició respecte a la versió original. També ha quedat pendent que usuaris fora de l'equip IT participin en el recull de requisits.

Els requeriments futurs no s'han pogut implementar, tal com es va preveure a la planificació. Pel que fa als requisits no funcionals, s'han pogut assolir tots.

Personalment, aquest treball m'ha servit per consolidar els conceptes apresos durant el Grau d'Enginyeria Informàtica, i adquirir nous coneixements en el desenvolupament d'una aplicació que utilitza noves tecnologies, que segueixen les tendències del sector i estan en evolució contínua. El projecte m'ha servit especialment per aprendre la part de desenvolupament web, ja que és un àmbit nou per a mi i considero que el resultat obtingut és molt positiu.

Tot i no haver pogut desenvolupar totes les funcionalitats que m'hagués agradat, estic satisfeta del resultat.

Pel que fa a la cooperativa, l'aplicació implementada permetrà empoderar als equips responsables de les comunicacions dels canvis que han de patir aquestes. També permetrà estalviar temps i esforços de coordinació.

Finalment, el codi l'he implementat amb una llicència de Codi lliure, per tan altres equips que utilitzin les plantilles Mako podrien aprofitar o adaptar-la per al seu ús.

Capítol 12

Treball futur

L'eina implementada durant el projecte és una primera versió funcional, però ha faltat el desenvolupament d'alguns requeriments desitjables, que serien els primers a implementar en la següent fase de desenvolupament, juntament amb els requisits ja identificats com a futurs. En aquesta propera fase també serà important incorporar la visió d'usuaris fora de l'equip IT, i desenvolupar els requisits que hagin detectat.

Més enllà d'aquests, el següent treball futur és adaptar l'UI-QMako per a poder editar documents adjunts (definits també amb plantilles Mako), llegint-ne la versió actual i podent pujar els canvis realitzats. Per a aquesta adaptació cal implementar la lectura i escriptura de fitxers al servidor.

Una altra funcionalitat que no ha sigut possible desenvolupar dins del marc del Projecte de final de grau, i que és molt important per a la cooperativa, és poder integrar el sistema de traduccions que utilitza Som Energia a l'UI-QMako. En un futur pròxim es vol poder emetre les comunicacions (correus electrònics i adjunts) en els 4 idiomes oficials de l'Estat Espanyol i fa necessari integrar l'eina de modificació dels texts amb un sistema àgil i eficaç de traducció.

Bibliografia

[Bayer 2021a] Michael Bayer. *Mako Templates for Python*, (Accessed: July 2021), 2021. Available at https://www.makotemplates.org/. (Cited on pages 2 and 11.)

[Bayer 2021b] Michael Bayer. *Syntax*, (Accessed: July 2021), 2021. Available at https://docs.makotemplates.org/en/latest/syntax.html. (Cited on page 12.)

[Beck 2002] Kent Beck. est-Driven Development by Example. 2002. (Cited on page 16.)

[Encode 2021a] Encode. *Django REST framework*, (Accessed: July 2021), 2021. Available at https://www.django-rest-framework.org/. (Cited on page 27.)

[Encode 2021b] Encode. *Starlette*, (Accessed: may 2021), 2021. Available at https://www.starlette.io/. (Cited on page 31.)

[Facebook 2021a] Facebook. *Context*, (Accessed: july 2021), 2021. Available at https://reactjs.org/docs/context.html. (Cited on page 59.)

[Facebook 2021b] Facebook. *React*, (Accessed: July 2021), 2021. Available at https://reactjs.org/. (Cited on pages 32 and 33.)

[Foundation 2021] Django Software Foundation. *The Django template language*, (Accessed: July 2021), 2005-2021. Available at https://docs.djangoproject.com/en/3.2/ref/templates/language/. (Cited on page 11.)

[Google 2021] Google. *Material Design*, (Accessed: may 2021), 2021. Available at https://github.com/sheaivey/react-axios. (Cited on page 33.)

[JiHong88 2021] JiHong88. *Sun Editor*, (Accessed: July 2021), 2021. Available at http://suneditor.com/sample/index.html. (Cited on page 34.)

[Kinev 2007] Alexey Kinev. *peewee-async*, (Accessed: July 2021), 2007. Available at https://peewee-async.readthedocs.io/en/latest/. (Cited on page 31.)

[Krekel 2020a] Holger Krekel. *Pytest fixtures: explicit, modular, scalable*, (Accessed: may 2021), 2015-2020. Available at https://docs.pytest.org/en/6.2.x/fixture.html. (Cited on page 31.)

[Krekel 2020b] Holger Krekel. *pytest: helps you write better programs*, (Accessed: may 2021), 2015-2020. Available at https://docs.pytest.org/en/6.2.x/. (Cited on page 31.)

[mkhstar 2007] mkhstar. *suneditor-react*, (Accessed: July 2021), 2007. Available at https://github.com/mkhstar/suneditor-react. (Cited on page 34.)

[mui org 2021] mui org. *Material-UI*, (Accessed: may 2021), 2021. Available at https://material-ui.com/. (Cited on page 33.)

[npm 2021] npm. *npm Docs*, (Accessed: may 2021), 2021. Available at https://docs.npmjs.com/about-npm. (Cited on page 55.)

[OpenapiInitiative 2007] OpenapiInitiative. *OpenAPI*, (Accessed: July 2021), 2007. Available at https://github.com/OAI/OpenAPI-Specification. (Cited on page 29.)

[Org 2007] JSON Schema Org. *JSON Schema*, (Accessed: July 2021), 2007. Available at https://json-schema.org/. (Cited on page 29.)

[Pallets 2007a] Pallets. *Flask*, (Accessed: July 2021), 2007. Available at https://flask.palletsprojects.com/en/2.0.x/. (Cited on page 27.)

[Pallets 2007b] Pallets. *Jinja*, (Accessed: July 2021), 2007. Available at https://jinja.palletsprojects.com/en/3.0.x/. (Cited on page 11.)

[Parecki 2021] Aaron Parecki. *OAuth 2.0*, (Accessed: July 2021), 2021. Available at https://oauth.net/2/. (Cited on page 32.)

[python poetry 2021] python poetry. *Poetry*, (Accessed: may 2021), 2018-2021. Available at https://python-poetry.org/docs/. (Cited on page 51.)

[Sandy 2021] James Sandy. *Choosing between Django, Flask, and FastAPI*, (Accessed: July 2021), 2021. Available at https://www.section.io/engineering-education/choosing-between-django-flask-and-fastapi/. (Cited on page 28.)

[sheaivey 2021] sheaivey. *React-Axios*, (Accessed: July 2021), 2021. Available at https://github.com/sheaivey/react-axios. (Cited on page 33.)

[Som Energia 2021] Equip IT Som Energia. *somenergia-oomakotest*, (Accessed: July 2021), 2015-2021. Available at https://github.com/Som-Energia/somenergia-oomakotest. (Cited on page 15.)

[tiangolo 021] tiangolo. *FastAPI*, (Accessed: July 2021). Available at https://fastapi.tiangolo.com/. (Cited on pages 27 and 30.)

[tiangolo 2007] tiangolo. *FastAPI Fists steps*, (Accessed: July 2021), 2007. Available at https://fastapi.tiangolo.com/tutorial/first-steps/. (Cited on page 29.)

[Wikipedia 2021] Wikipedia. *Multiton pattern*, (Accessed: july 2021), 2021. Available at https://en.wikipedia.org/wiki/Multiton_pattern. (Cited on page 49.)

Annex I: Planificació final detallada

Per a la planificació de les tasques a realitzar he utilitzat la eina *Trello*, on cada columna representa una Ronda. La estimació del temps es mostra en unitats de *mig dia* (considerant una jornada laboral de 8 h, un mig dia correspon a 4 h de feina). Entre parèntesis es mostra la dedicació real/estimada de cada tasca. En general, la planificació estimada s'ha ajustat a la real.

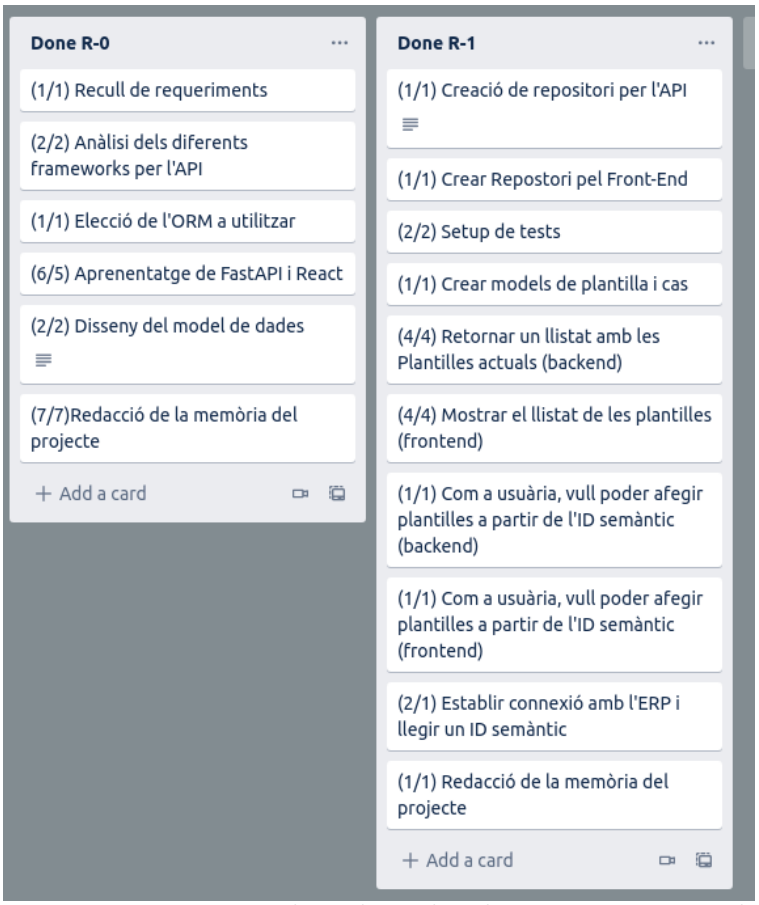

Figura 1: Tasques realitzades a les dues primeres Rondes

Figura 2: Tasques realitzades a les Rondes 2 i 3.

Figura 3: Tasques realitzades a les Rondes 4 i 5.

Annex II: Documentació de l'API

Accedint a la ruta */docs* de l'API es pot consultar la documentació de l'API en format web:

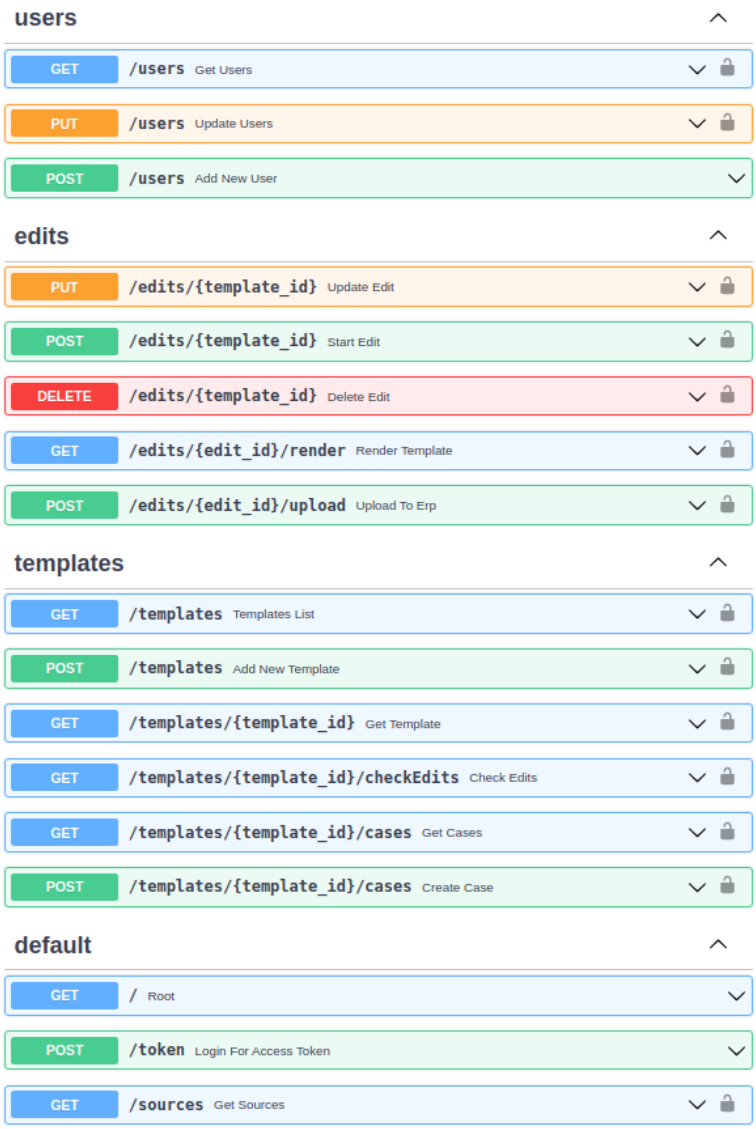

Figura 4: Vista dels endpoints accedint a la ruta */docs* de l'API.

Accedint a la ruta */openapi.json* el JSON Schema d'aquesta:


```
{
  "openapi": "3.0.2",
  "info": { "title": "FastAPI", "version": "0.1.0" },
  "paths": {
    "/users": {
      "get": {
        "tags": ["users"],
        "summary": "Get Users",
        "operationId": "get_users_users_get",
        "responses": {
          "200": {
            "description": "Successful Response",
            "content": { "application/json": { "schema": {} } }
          }
        },
        "security": [{ "OAuth2PasswordBearer": [] }]
      },
      "put": {
        "tags": ["users"],
        "summary": "Update Users",
        "operationId": "update_users_users_put",
        "requestBody": {
          "content": {
            "application/json": {
              "schema": { "$ref": "#/components/schemas/UserInPost" }
            }
          },
          "required": true
        },
        "responses": {
          "200": {
            "description": "Successful Response",
            "content": { "application/json": { "schema": {} } }
          },
          "422": {
            "description": "Validation Error",
            "content": {
              "application/json": {
                "schema": { "$ref": "#/components/schemas/HTTPValidationError" }
              }
            }
          }
        },
        "security": [{ "OAuth2PasswordBearer": [] }]
      },
      "post": {
        "tags": ["users"],
        "summary": "Add New User",
        "operationId": "add_new_user_users_post",
        "requestBody": {
          "content": {
            "application/x-www-form-urlencoded": {
              "schema": {
                "$ref": "#/components/schemas/Body_add_new_user_users_post"
              }
            }
          },
```

```
          "required": true
        },
        "responses": {
          "200": {
            "description": "Successful Response",
            "content": { "application/json": { "schema": {} } }
          },
          "422": {
            "description": "Validation Error",
            "content": {
              "application/json": {
                "schema": { "$ref": "#/components/schemas/HTTPValidationError" }
              }
            }
          }
        }
      }
    },
    "/edits/{template_id}": {
      "put": {
        "tags": ["edits"],
        "summary": "Update Edit",
        "operationId": "update_edit_edits__template_id__put",
        "parameters": [
          {
            "required": true,
            "schema": { "title": "Template Id", "type": "integer" },
            "name": "template_id",
            "in": "path"
          }
        ],
        "requestBody": {
          "content": {
            "application/json": {
              "schema": { "$ref": "#/components/schemas/RawEdit" }
            }
          },
          "required": true
        },
        "responses": {
          "200": {
            "description": "Successful Response",
            "content": { "application/json": { "schema": {} } }
          },
          "422": {
            "description": "Validation Error",
            "content": {
              "application/json": {
                "schema": { "$ref": "#/components/schemas/HTTPValidationError" }
              }
            }
          }
        },
        "security": [{ "OAuth2PasswordBearer": [] }]
      },
      "post": {
        "tags": ["edits"],
        "summary": "Start Edit",
        "operationId": "start_edit_edits__template_id__post",
        "parameters": [
          {
```

```
          "required": true,
          "schema": { "title": "Template Id", "type": "integer" },
          "name": "template_id",
          "in": "path"
        }
      ],
      "responses": {
        "200": {
          "description": "Successful Response",
          "content": { "application/json": { "schema": {} } }
        },
        "422": {
          "description": "Validation Error",
          "content": {
            "application/json": {
              "schema": { "$ref": "#/components/schemas/HTTPValidationError" }
            }
          }
        }
      },
      "security": [{ "OAuth2PasswordBearer": [] }]
    },
    "delete": {
      "tags": ["edits"],
      "summary": "Delete Edit",
      "operationId": "delete_edit_edits__template_id__delete",
      "parameters": [
        {
          "required": true,
          "schema": { "title": "Template Id", "type": "integer" },
          "name": "template_id",
          "in": "path"
        }
      ],
      "responses": {
        "200": {
          "description": "Successful Response",
          "content": { "application/json": { "schema": {} } }
        },
        "422": {
          "description": "Validation Error",
          "content": {
            "application/json": {
              "schema": { "$ref": "#/components/schemas/HTTPValidationError" }
            }
          }
        }
      },
      "security": [{ "OAuth2PasswordBearer": [] }]
    }
  },
  "/edits/{edit_id}/render": {
    "get": {
      "tags": ["edits"],
      "summary": "Render Template",
      "operationId": "render_template_edits__edit_id__render_get",
      "parameters": [
        {
          "required": true,
          "schema": { "title": "Edit Id", "type": "integer" },
          "name": "edit_id",
```

```
            "in": "path"
          },
          {
            "required": true,
            "schema": { "title": "Case Id", "type": "integer" },
            "name": "case_id",
            "in": "query"
          }
        ],
        "responses": {
          "200": {
            "description": "Successful Response",
            "content": { "application/json": { "schema": {} } }
          },
          "422": {
            "description": "Validation Error",
            "content": {
              "application/json": {
                "schema": { "$ref": "#/components/schemas/HTTPValidationError" }
              }
            }
          }
        },
        "security": [{ "OAuth2PasswordBearer": [] }]
      }
    },
    "/edits/{edit_id}/upload": {
      "post": {
        "tags": ["edits"],
        "summary": "Upload To Erp",
        "operationId": "upload_to_erp_edits__edit_id__upload_post",
        "parameters": [
          {
            "required": true,
            "schema": { "title": "Edit Id", "type": "integer" },
            "name": "edit_id",
            "in": "path"
          },
          {
            "required": true,
            "schema": { "title": "Source", "type": "string" },
            "name": "source",
            "in": "query"
          }
        ],
        "responses": {
          "200": {
            "description": "Successful Response",
            "content": { "application/json": { "schema": {} } }
          },
          "422": {
            "description": "Validation Error",
            "content": {
              "application/json": {
                "schema": { "$ref": "#/components/schemas/HTTPValidationError" }
              }
            }
          }
        },
        "security": [{ "OAuth2PasswordBearer": [] }]
      }
    },
```

```
    "/templates": {
      "get": {
        "tags": ["templates"],
        "summary": "Templates List",
        "operationId": "templates_list_templates_get",
        "responses": {
          "200": {
            "description": "Successful Response",
            "content": { "application/json": { "schema": {} } }
          }
        },
        "security": [{ "OAuth2PasswordBearer": [] }]
      },
      "post": {
        "tags": ["templates"],
        "summary": "Add New Template",
        "operationId": "add_new_template_templates_post",
        "requestBody": {
          "content": {
            "application/x-www-form-urlencoded": {
              "schema": {
                "$ref":
                  "#/components/schemas/Body_add_new_template_templates_post"
              }
            }
          },
          "required": true
        },
        "responses": {
          "200": {
            "description": "Successful Response",
            "content": { "application/json": { "schema": {} } }
          },
          "422": {
            "description": "Validation Error",
            "content": {
              "application/json": {
                "schema": { "$ref": "#/components/schemas/HTTPValidationError" }
              }
            }
          }
        },
        "security": [{ "OAuth2PasswordBearer": [] }]
      }
    },
    "/templates/{template_id}": {
      "get": {
        "tags": ["templates"],
        "summary": "Get Template",
        "operationId": "get_template_templates__template_id__get",
        "parameters": [
          {
            "required": true,
            "schema": { "title": "Template Id", "type": "integer" },
            "name": "template_id",
            "in": "path"
          }
        ],
        "responses": {
          "200": {
            "description": "Successful Response",
```

```
            "content": { "application/json": { "schema": {} } }
          },
          "422": {
            "description": "Validation Error",
            "content": {
              "application/json": {
                "schema": { "$ref": "#/components/schemas/HTTPValidationError" }
              }
            }
          }
        },
        "security": [{ "OAuth2PasswordBearer": [] }]
      }
    },
    "/templates/{template_id}/checkEdits": {
      "get": {
        "tags": ["templates"],
        "summary": "Check Edits",
        "operationId": "check_edits_templates__template_id__checkEdits_get",
        "parameters": [
          {
            "required": true,
            "schema": { "title": "Template Id", "type": "integer" },
            "name": "template_id",
            "in": "path"
          }
        ],
        "responses": {
          "200": {
            "description": "Successful Response",
            "content": { "application/json": { "schema": {} } }
          },
          "422": {
            "description": "Validation Error",
            "content": {
              "application/json": {
                "schema": { "$ref": "#/components/schemas/HTTPValidationError" }
              }
            }
          }
        },
        "security": [{ "OAuth2PasswordBearer": [] }]
      }
    },
    "/templates/{template_id}/cases": {
      "get": {
        "tags": ["templates"],
        "summary": "Get Cases",
        "operationId": "get_cases_templates__template_id__cases_get",
        "parameters": [
          {
            "required": true,
            "schema": { "title": "Template Id", "type": "integer" },
            "name": "template_id",
            "in": "path"
          }
        ],
        "responses": {
          "200": {
            "description": "Successful Response",
            "content": { "application/json": { "schema": {} } }
```

```
        },
        "422": {
          "description": "Validation Error",
          "content": {
            "application/json": {
              "schema": { "$ref": "#/components/schemas/HTTPValidationError" }
            }
          }
        }
      },
      "security": [{ "OAuth2PasswordBearer": [] }]
    },
    "post": {
      "tags": ["templates"],
      "summary": "Create Case",
      "operationId": "create_case_templates__template_id__cases_post",
      "parameters": [
        {
          "required": true,
          "schema": { "title": "Template Id", "type": "integer" },
          "name": "template_id",
          "in": "path"
        }
      ],
      "requestBody": {
        "content": {
          "application/x-www-form-urlencoded": {
            "schema": {
              "$ref":
                "#/components/schemas/
                   Body_create_case_templates__template_id__cases_post"
            }
          }
        },
        "required": true
      },
      "responses": {
        "200": {
          "description": "Successful Response",
          "content": { "application/json": { "schema": {} } }
        },
        "422": {
          "description": "Validation Error",
          "content": {
            "application/json": {
              "schema": { "$ref": "#/components/schemas/HTTPValidationError" }
            }
          }
        }
      },
      "security": [{ "OAuth2PasswordBearer": [] }]
    }
  },
  "/": {
    "get": {
      "summary": "Root",
      "operationId": "root__get",
      "responses": {
        "200": {
          "description": "Successful Response",
          "content": { "application/json": { "schema": {} } }
```

```
        }
      }
    }
  },
  "/token": {
    "post": {
      "summary": "Login For Access Token",
      "operationId": "login_for_access_token_token_post",
      "requestBody": {
        "content": {
          "application/x-www-form-urlencoded": {
            "schema": {
              "$ref":
                 "#/components/schemas/Body_login_for_access_token_token_post"
            }
          }
        },
        "required": true
      },
      "responses": {
        "200": {
          "description": "Successful Response",
          "content": {
            "application/json": {
              "schema": { "$ref": "#/components/schemas/TokenInPost" }
            }
          }
        },
        "422": {
          "description": "Validation Error",
          "content": {
            "application/json": {
              "schema": { "$ref": "#/components/schemas/HTTPValidationError" }
            }
          }
        }
      }
    }
  },
  "/sources": {
    "get": {
      "summary": "Get Sources",
      "operationId": "get_sources_sources_get",
      "responses": {
        "200": {
          "description": "Successful Response",
          "content": { "application/json": { "schema": {} } }
        }
      },
      "security": [{ "OAuth2PasswordBearer": [] }]
    }
  }
},
"components": {
  "schemas": {
    "Body_add_new_template_templates_post": {
      "title": "Body_add_new_template_templates_post",
      "required": ["xml_id"],
      "type": "object",
      "properties": {
        "xml_id": {
          "title": "Xml Id",
```

```
        "pattern": ".+\\..+",
        "type": "string"
      }
    }
  },
  "Body_add_new_user_users_post": {
    "title": "Body_add_new_user_users_post",
    "required": ["username", "password"],
    "type": "object",
    "properties": {
      "username": { "title": "Username", "type": "string" },
      "password": { "title": "Password", "type": "string" }
    }
  },
  "Body_create_case_templates__template_id__cases_post": {
    "title": "Body_create_case_templates__template_id__cases_post",
    "required": ["case_name", "case_id"],
    "type": "object",
    "properties": {
      "case_name": { "title": "Case Name", "type": "string" },
      "case_id": {
        "title": "Case Id",
        "pattern": ".+\\..+|[1-9][0-9]?",
        "type": "string"
      }
    }
  },
  "Body_login_for_access_token_token_post": {
    "title": "Body_login_for_access_token_token_post",
    "required": ["username", "password"],
    "type": "object",
    "properties": {
      "grant_type": {
        "title": "Grant Type",
        "pattern": "password",
        "type": "string"
      },
      "username": { "title": "Username", "type": "string" },
      "password": { "title": "Password", "type": "string" },
      "scope": { "title": "Scope", "type": "string", "default": "" },
      "client_id": { "title": "Client Id", "type": "string" },
      "client_secret": { "title": "Client Secret", "type": "string" }
    }
  },
  "HTTPValidationError": {
    "title": "HTTPValidationError",
    "type": "object",
    "properties": {
      "detail": {
        "title": "Detail",
        "type": "array",
        "items": { "$ref": "#/components/schemas/ValidationError" }
      }
    }
  },
  "RawEdit": {
    "title": "RawEdit",
    "type": "object",
    "properties": {
      "by_type": { "title": "By Type", "type": "string" },
      "def_body_text": { "title": "Def Body Text", "type": "string" },
```

```
          "headers": { "title": "Headers", "type": "string" }
        }
      },
      "TokenInPost": {
        "title": "TokenInPost",
        "required": ["access_token"],
        "type": "object",
        "properties": {
          "access_token": { "title": "Access Token", "type": "string" },
          "token_type": {
            "title": "Token Type",
            "type": "string",
            "default": "bearer"
          }
        }
      },
      "UserCategory": {
        "title": "UserCategory",
        "enum": ["basic_user", "html_user", "python_user", "admin"],
        "type": "string",
        "description": "An enumeration."
      },
      "UserInPost": {
        "title": "UserInPost",
        "required": ["id", "username"],
        "type": "object",
        "properties": {
          "id": { "title": "Id", "type": "integer" },
          "username": { "title": "Username", "type": "string" },
          "disabled": { "title": "Disabled", "type": "boolean" },
          "category": {
            "allOf": [{ "$ref": "#/components/schemas/UserCategory" }],
            "default": "basic_user"
          }
        }
      },
      "ValidationError": {
        "title": "ValidationError",
        "required": ["loc", "msg", "type"],
        "type": "object",
        "properties": {
          "loc": {
            "title": "Location",
            "type": "array",
            "items": { "type": "string" }
          },
          "msg": { "title": "Message", "type": "string" },
          "type": { "title": "Error Type", "type": "string" }
        }
      }
    },
    "securitySchemes": {
      "OAuth2PasswordBearer": {
        "type": "oauth2",
        "flows": { "password": { "scopes": {}, "tokenUrl": "token" } }
      }
    }
  }
}
```


Listing 11: JSON Schema de l'APi