Universitat Oberta de Catalunya

uoc.edu

Gestió treballadors d'empresa HR ORIGIN

Carlos Liarte Navarro
Grau d'Enginyeria Informàtica
Desenvolupament web

Consultor/a Vicenç Font Sagrista
Professor/a responsable Santi Caballe Llobet

Juny 2022

FITXA DEL TREBALL FINAL

Títol del treball:	*Gestió plantilla d'empresa HR ORIGIN*
Nom de l'autor:	*Carlos Liarte Navarro*
Nom del consultor/a:	*Vicenç Font Sagrista*
Nom del PRA:	*Santi Caballe Llobet*
Data de lliurament (mm/aaaa):	*06/2022*
Titulació o programa:	*Grau d'Enginyeria Informàtica*
Àrea del Treball Final:	*Desenvolupament web*
Idioma del treball:	*Català*
Paraules clau	*Recursos humans, Java, Spring Boot*

Resum del Treball (màxim 250 paraules): *Amb la finalitat, context d'aplicació, metodologia, resultats i conclusions del treball*

La finalitat del projecte és atorgar a les empreses una eina que faciliti les tasques de gestió d'empleats així com proporcionar la instrumentació necessària per reduir els temps de cerca i contractació de nous empleats.

Al context actual existeixen moltes empreses que fan ús de fulls de càlcul o correus electrònics per a tenir un seguiment de les candidatures i amb el projecte Origin es vol unificar tota la informació en una mateixa plataforma on puguin interactuar tant els recursos humans de l'empresa com els candidats amb total visibilitat i transparència dels processos de selecció de candidats.

El desenvolupament de la plataforma es basa en una metodologia àgil on cada mòdul de l'aplicació s'ha dut a terme mitjançant cicles de desenvolupament de programari des de la mateixa definició del requisit fins a les proves pertinents per tal d'afavorir la cobertura i seguretat del codi.

Com a resultat s'ha obtingut una plataforma de gestió de plantilles i selecció de personal robusta amb una certa traçabilitat dels processos de selecció mitjançant notificacions per correu electrònic. A més, la mateixa empresa podrà crear la seva pròpia base de dades de candidats per a futures posicions obertes dins de la companyia.

En conclusió, el candidat assolirà major visibilitat i comunicació amb el personal encarregat del procés de selecció i la persona encarregada de gestionar la posició oberta tindrà tota la informació del personal i candidats centralitzada en un mateix lloc garantint la seguretat i protecció de les dades.

Abstract (in English, 250 words or less):

The aim of the project is to provide companies with a tool that facilitates employee management tasks as well as providing the necessary tools to reduce the search and hiring time for new employees.

In the current context there are many companies that use spreadsheets or emails to keep track of applications and the Origin project wants to unify all the information on a single platform where both human resources of the company can interact as candidates with full visibility and transparency of candidate selection processes.

The development of the platform is based on an agile methodology where each module of the application has been carried out through software development cycles from the very definition of the requirement to the relevant tests in order to favor the coverage and code security.

As a result, a robust template management and selection platform has been achieved with some traceability of selection processes through email notifications. In addition, the same company will be able to create its own database of candidates for future open positions within the company.

In conclusion, the candidate will achieve greater visibility and communication with the staff in charge of the selection process and the person in charge of managing the open position will have all the information of the staff and candidates centralized in one place ensuring the security and protection of data.

Índex

Llista de figures

Llista de taules

1. Introducció

1.1. Context i justificació del Treball

Actualment existeixen diverses empreses que tenen bastants problemes per gestionar les plantilles de treballadors d'aquestes a nivell de cobrir baixes per malaltia o vacances d'una manera àgil sense necessitat de d'utilitzar fulls de càlcul o muntanyes de paper que acaben generant frustració, estrès i una baixa productivitat dels responsables de recursos humans.

Per aquest motiu aquesta aplicació està orientada a solucionar la problemàtica explicada anteriorment. Com es pot observar en aquests temps on la població activa ha patit bastants baixes per malaltia incrementant d'aquesta manera la necessitat d'incorporar professionals a les plantilles de les empreses d'una manera ràpida i àgil.

Avui dia els responsables de recursos humans normalment solucionen aquesta problemàtica mitjançant l'ús de correus electrònics per contactar contínuament amb els candidats per informar dels processos de selecció a més de fer ús de fulls de càlcul per conèixer l'estat de la plantilla.

Per tant, l'objectiu del desenvolupament d'aquest programari és obtenir una aplicació on els responsables del departament de recursos humans de les empreses amb una alta incidència de baixes o de rotacions a la plantilla.

D'aquesta manera podran treballar d'una manera efectiva amb total seguretat de no tenir incongruències a les dades recol·lectades als diferents fulls de càlcul utilitzat en aquests processos i, per tant, tindran una visibilitat constant de l'estat de les plantilles per poder fer front a aquestes complicacions.

1.2. Objectius del Treball

L'objectiu principal del treball és obtenir una aplicació web on el departament de recursos pugui saber en tot moment l'estat de la plantilla en l'àmbit nivell de vacances com de baixes per malaltia i així d'aquesta manera suplir aquestes persones si fos necessari de la manera més ràpida i eficient.

I d'aquesta manera podrem obtenir els objectius generals del projecte que són:

- Automatitzar la gestió de baixes d'una plantilla i agilitzar el treball dels responsables de recursos humans de l'empresa.
- Mecanitzar els processos de candidatures de l'empresa.
- Donar visibilitat de l'estat de la plantilla pels diferents departaments de l'empresa.
- Desenvolupar una aplicació que fàcil de mantenir amb una usabilitat multiplataforma en diferents idiomes.

Per arribar a aquests objectius generals podríem desglossar-los en objectius per mòduls i, per tant, seran els següents:

- El primer mòdul seria la gestió dels empleats per part dels mànagers de l'empresa els quals tenen assignats i d'aquesta manera aquests seran els encarregats d'aprovar les peticions de vacances per part dels empleats.
- El segon mòdul serà que els empleats podran avisar que estan de baixa mitjançant l'aplicació i d'aquesta manera el departament de recursos humans serà notificat

perquè aquests puguin cobrir el lloc de treball a través d'una borsa de candidats preseleccionats amb anterioritat.

- El tercer mòdul fa referència a la creació de llistes de candidats per cadascun dels departaments per poder anar realitzant entrevistes i ponderant els futurs candidats a ocupar un lloc de treball a l'empresa o suplir una baixa durant un temps parcial o complet.

1.3. Enfocament i mètode seguit

Per dur a terme el treball desenvoluparem un producte nou afitat als requeriments obtinguts, ja que no trobem cap programari de codi obert que compleixi els requeriments tant funcionals com no funcionals.

D'aquesta manera podrem realitzar un programari a mesura per complir els objectius separats per mòduls i, per tant, es podran afegir nous mòduls en un futur si es consideren noves funcionalitats per part de les empreses que utilitzen l'aplicació duta a terme en aquest treball.

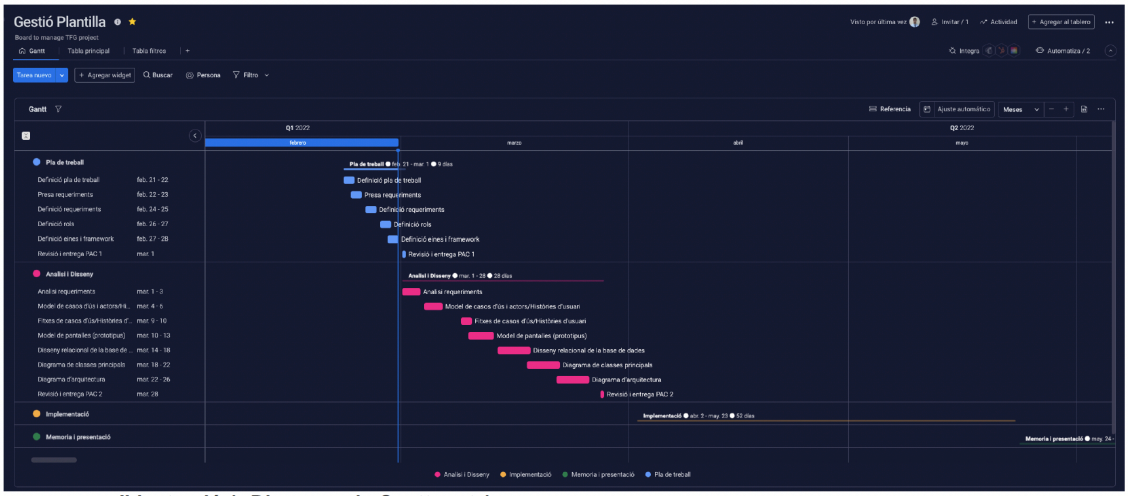
Una de les estratègies d'implantació d'aquest programari a les empreses és que els treballadors i els candidats s'aniran creant mitjançant un formulari al navegador web.

Per treballar d'una manera àgil es portaran a cap planificacions per cada entrega per dur a terme cadascuna de les tasques arran de la planificació inicial de tasques.

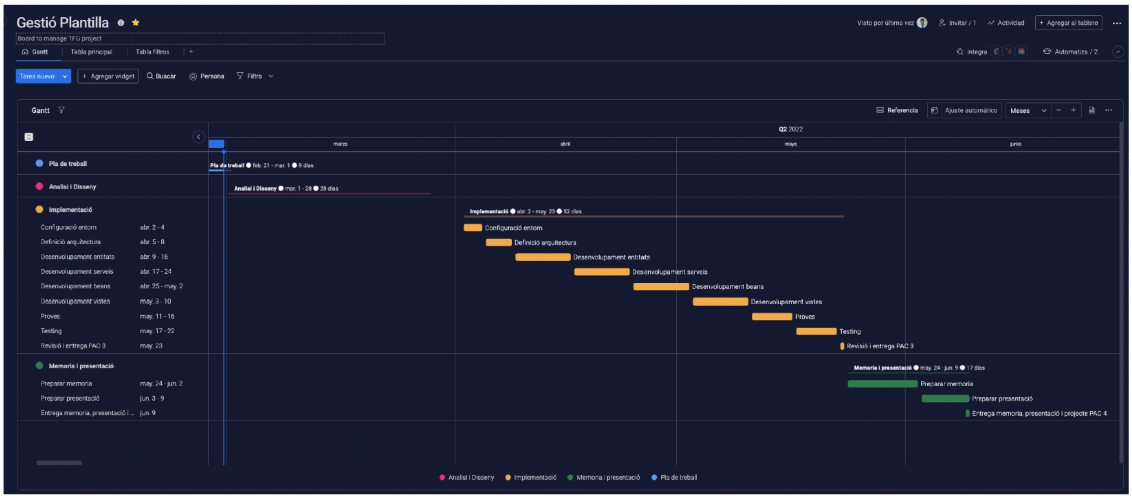
1.4. Planificació del Treball

Per a realitzar la planificació del treball s'ha escollit l'eina Monday[1] on hem definit una primera taula amb la definició de les tasques d'acord amb les fites d'entregues per arribar a l'objectiu final d'entregar una primera versió del producte de gestió de plantilles.

Un cop portada a cap la taula de tasques, l'eina és capaç de traduir-la en un diagrama de Gantt bastant intuïtiu.



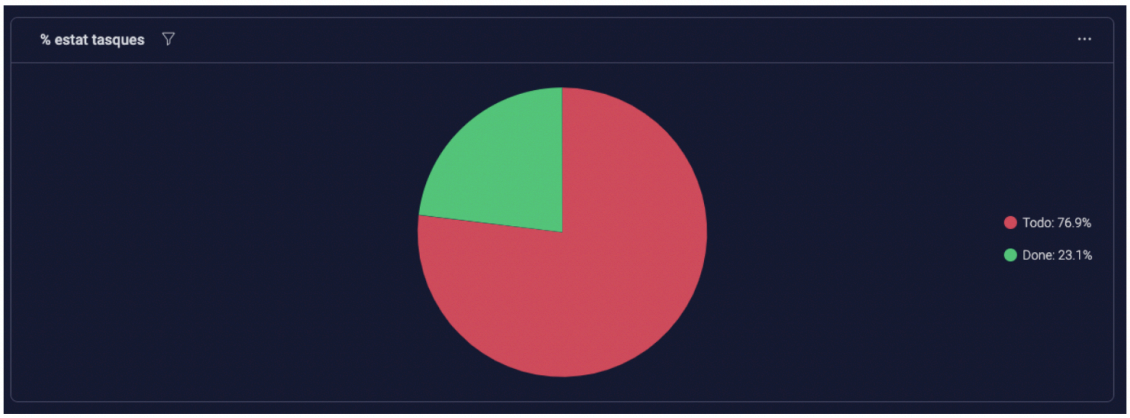
Il·lustració 1: Diagrama de Gantt part 1



Il·lustració 2: Diagrama de Gantt part 2

També s'han afegit algunes mètriques per conèixer l'estat de les tasques:

Il·lustració 3: Diagrama numero de tasques per estat



Il·lustració 4: Diagrama % d'estat de les tasques

1.5. Tecnologies utilitzades

- Spring Boot 2.6.3[4]
- Maven 3.8.4[5]
- PrimeFaces 8.0.0[6]
- Java SE Development Kit 11[7]
- IntelliJ v2021.3.2[8]
- PostgreSQL v12[9]
- Git[10]
- Github[11]
- Heroku[12]

1.6. Breu sumari de productes obtinguts

Els productes obtinguts al llarg del desenvolupament del projecte són els següents:

- Aplicació web
- Codi font
- Memòria final del projecte
- Presentació del projecte

1.7. Breu descripció dels altres capítols de la memòria

A continuació es mostrarà una breu descripció dels diferents capítols que formen la memòria basada en els 4 grans mòduls:

Introducció:

En aquest apartat es realitza una introducció al treball enfocada a la part inicial del projecte on es defineixen alguns aspectes importants com el context i situació la qual es troba el projecte just abans de començar.

A més es redacten els objectius als quals es volen arribar un cop finalitzi el projecte tot i que a vegades no s'aconsegueixin complir perquè el projecte és molt ambiciós.

A continuació s'explica com serà la planificació del projecte estructurada per mòduls en funció dels diferents lliurables que es portaran a cap durant el termini del semestre així com les tecnologies desitjades per desenvolupar l'aplicació web.

Anàlisi de requisits:

Aquest capítol està enfocat en la mateixa definició de tots els requisits que necessita complir l'aplicació definida pels interessats que en aquesta ocasió seria el mateix desenvolupador del projecte. A més, existeixen uns altres requisits que són els no funcionals perquè seran aquells que no realitza accions l'usuari amb la plataforma, però que són igual d'importants.

Per concloure aquest capítol també defineix els rols dels usuaris que hi trobarem a l'aplicació i que en funció d'aquests rols els usuaris tindran diferents permisos per interactuar amb la plataforma.

Disseny de l'aplicació:

En aquesta aplicació s'han emprat diferents tipus de diagrames per definir per exemple la base de dades així les diferents capes amb les seves respectives interfícies i *beans*. D'aquesta manera, s'aconsegueix tenir una base abans de començar a desenvolupar el codi pertinent tot i que a vegades existeixin canvis respecte de la definició inicial.

A més, es generen els prototipus de les diferents pantalles que compondran l'aplicació per facilitar les tasques de disseny un cop el desenvolupador comença a crear el codi referent a la capa de presentació.

Implementació de l'aplicació:

Aquest capítol explica tot el referent a la mateixa implementació de l'aplicació des de la configuració de l'entorn perquè funcioni correctament en local fins a les diferents proves realitzes per validar que el codi funciona tal com s'han redactat els diferents casos d'ús a l'apartat 3. A més, es detallen les diferents capes amb exemples reals de codi perquè es tingui una millor percepció de com s'ha executat el projecte.

Per concloure, trobem una explicació de com s'ha fet ús del mòdul de seguretat de Spring i es mostren parts del codi per millorar la comprensió del funcionament de la seguretat de la plataforma.

2. Anàlisi de requisits

Des del departament de recursos humans d'una empresa han de poder donar d'alta a la plataforma per a tots els candidats que hagin passat els processos previs de selecció per a realitzar substitucions. Per a cada candidat es volen les seves dades personals per contactar.

Per cada posició oberta hi podrem trobar diferents fases que seran definides de la següent manera:

- **Iniciada**: S'ha posat en marxa el procés de selecció de la nova posició oberta.
- **Entrevistant**: S'estan portant a cap les pertinents entrevistes als diferents candidats.
- **Pendent avaluació**: Els diferents integrants que assisteixen a les entrevistes estan valorant si algun candidat és correcte per la posició on opten a poder treballar.
- **Exitosa**: Els responsables de recursos humans han decidit voler incorporar un candidat.
- **Descartada**: La posició oberta ha estat descartada perquè no troben cap candidat, acord a la posició oberta.

La gestió de les llistes de candidats la podrà duar a terme un usuari amb el rol de recursos humans.

Els usuaris amb rol de recursos humans podran crear i gestionar les candidatures així com associar una posició oberta a un candidat. Aquestes candidatures contindran els comentaris associats al procés de selecció i a més tindran en tot moment un estat per informar al candidat en quin estat es troba el procés de selecció.

Els estats de la candidatura seran els següents:

- **Avaluant:** La candidatura s'està avaluant per part del departament de recursos humans de l'empresa.

- **Exitosa:** El candidat ha superat satisfactòriament el procés de selecció i ha estat seleccionat per formar part de l'empresa.

- **Descartat:** El candidat no ha superat el procés de selecció i, per tant, els responsables de recursos humans han descartat la seva candidatura per ocupar la posició oberta.

Les llistes de candidats seran ordenades per data de finalització de candidatura i, per tant, portaran un ordre com si es tractés d'una cua FIFO.

Cada candidat podrà accedir a la plataforma per veure l'estat de la seva candidatura i d'aquesta manera estar informat en tot moment de l'estat d'aquesta.

Els usuaris RRHH podran visualitzar quins treballadors estan actualment de baixa per poder organitzar la contractació de noves persones.

Els usuaris RRHH podran contactar amb els candidats mitjançant un correu electrònic per saber del seu interès a ocupar un lloc de treball fix o temporal a l'empresa.

Un cop realitzat el contacte amb el candidat llavors la llista de candidats s'actualitzarà per informar que s'ha realitzat comunicació per a evitar que s'enviïn més d'un correu a la mateixa persona.

Els empleats de l'empresa (RRHH, Mànager, Treballador) tindran dret a demanar vacances quan vulguin i per això cada treballador haurà de poder sol·licitar una petició de vacances o baixa al seu respectiu mànager. Per tant, el mànager haurà de ser capaç d'acceptar o denegar les vacances del treballador en qüestió.

Cada empleat de l'empresa (RRHH, Mànager, Treballador) haurà de poder gestionar les seves dades personals.

Els usuaris amb rol d'administrador podran gestionar els usuaris de l'aplicació així com els departaments i assignar aquests a departaments.

Per al cas dels departaments els usuaris amb rol de recursos humans assignaran les posicions obertes a un departament.

2.1. Definició de rols i permisos

Codi	Nom
R1	Admin
R2	RRHH
R3	Mànager
R4	Treballador
R5	Candidat

2.2. Requisits funcionals

Codi	Requisit	Admin	RRHH	Màn	Treb	Cand
RF1	Identificar-se al sistema	X	X	X	X	X
RF2	Sortir del sistema	X	X	X	X	X
RF3	Gestionar usuaris	X				
RF4	Gestionar candidats		X			
RF5	Visualitzar posicions obertes per departament		X			
RF6	Visualitzar absències		X			
RF7	Contactar candidats		X			
RF8	Gestionar peticions absències			X		
RF9	Sol·licitar absència				X	
RF10	Gestionar dades personals		X	X	X	
RF11	Gestionar candidatures		X			
RF12	Visualitzar estat candidatura		X			X
RF13	Canviar estat candidatura		X			
RF14	Assignar posició oberta a departament		X			
RF15	Gestionar departaments	X				
RF16	Gestionar posicions obertes		X			
RF17	Assignar candidat a posició oberta		X			

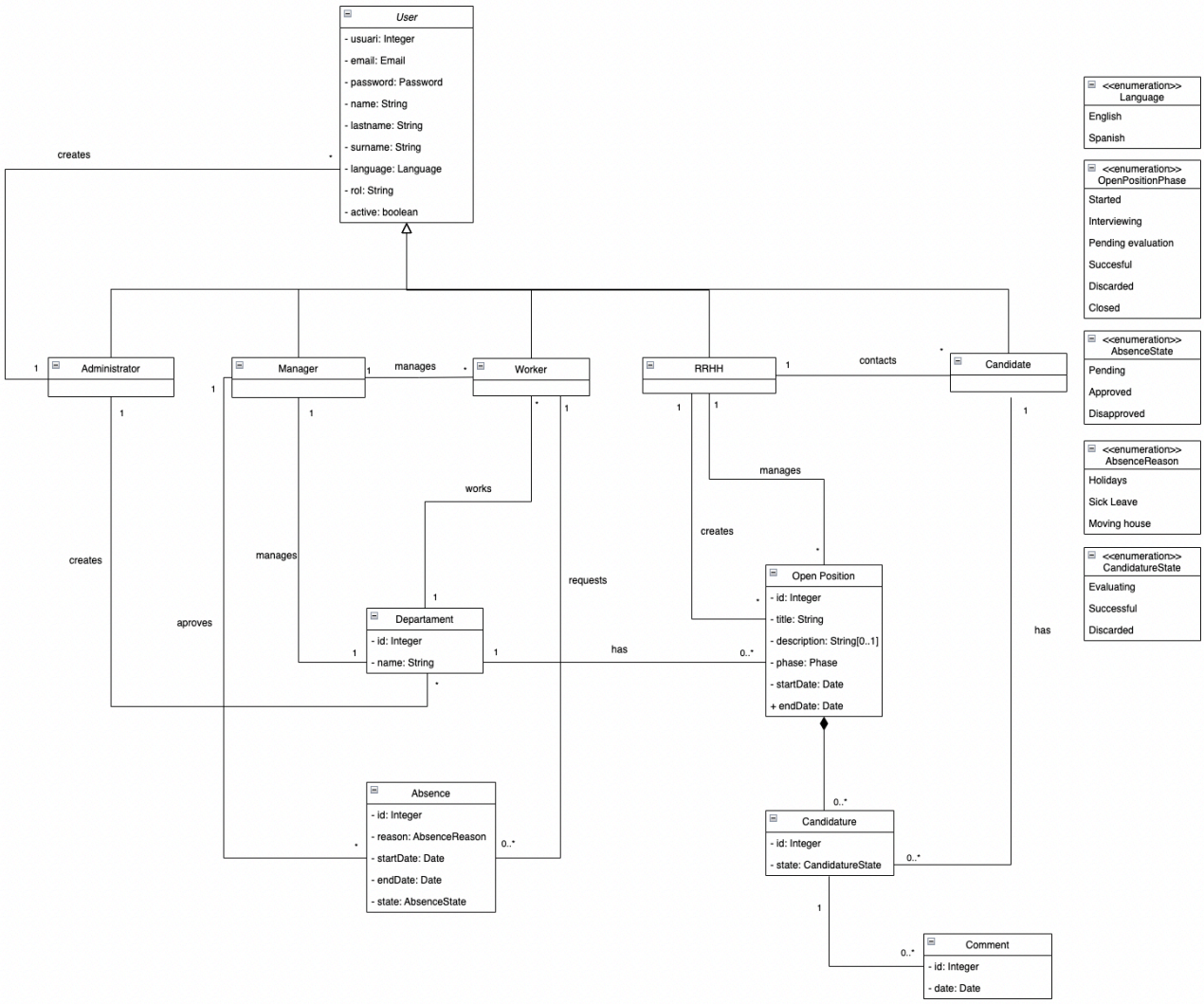
2.3. Requisits no funcionals

Codi	Requisit
RNF1	Aplicació multi-idioma (Castellà i anglès)
RNF2	Aplicació adaptativa per bona visualització tant a PC, mòbil o tableta.
RNF3	Encriptació de la contrasenya dels usuaris.

3. Disseny de l'aplicació

3.1. Diagrama UML

A partir de la definició obtinguda i de la definició dels requisits inicials es procedeix a la creació del disseny conceptual de l'aplicació a partir d'un diagrama UML com el següent on es mostren les entitats pertinents:



Il·lustració 5: Diagrama UML amb Draw.io[3]

3.2. Model relacional

User (usuari, email, password, name, lastname, surname, language, rol, active)

Administrator (usuari, email, password, name, lastname, surname, language, active)

Manager (usuari, email, password, name, lastname, surname, language, active, department)

{department} is a foreign key to Department

Worker (usuari, email, password, name, lastname, surname, language, active, department)

{department} is a foreign key to Department

RRHH (usuari, email, password, name, lastname, surname, language, active)

Candidate (usuari, nif, email, name, lastname, surname, email, language, active)

OpenPosition (idOpenPosition, title, description, phase, startDate, endDate, rrhh)

{rrhh} is a foreign key to RRHH

Candidature (id, candidate, openPosition, phase)

{candidate} is a foreign key to Candidate

{openPosition} is a foreign key to OpenPosition

Comment (id, date, candidature)

{candidature is a foreign key to Candidature}

Absence (idAbsence, reason, startDate, endDate, worker, state)

{worker} is a foreign key to Worker

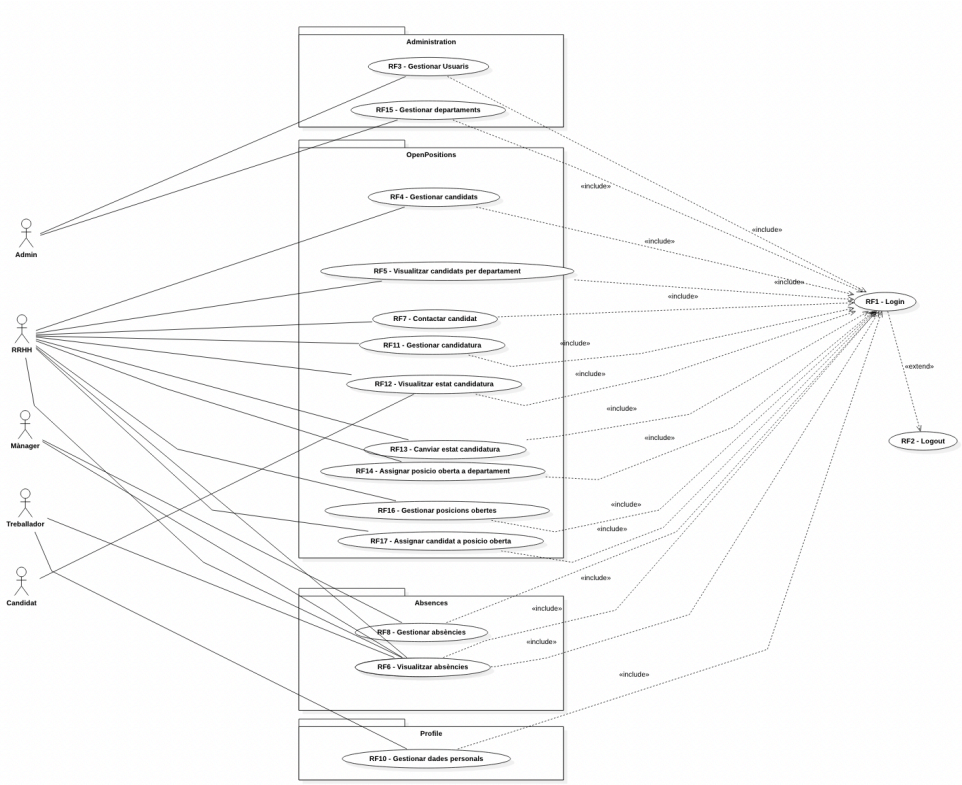
Department (id, name, manager)

{manager} is a foreign key to Manager

Els atributs subratllats seran aquells que siguin clau primària de l'entitat.

3.3. Casos d'ús

Diagrama de casos d'ús segons la definició dels requisits funcionals obtinguts a partir de la informació inicial.



Il·lustració 6: Diagrama casos d'ús amb StarUML[2]

3.4. Especificació de casos d'ús

A continuació s'expliquen en detall els diferents casos d'ús definits al diagrama anterior:

Cas d'ús: RF1 – Identificar-se a l'aplicació
- Actors: Administrador, RRHH, Mànager i Treballador.
- Descripció: L'usuari indica el nom i la contrasenya al formulari d'inici de sessió.
- Condició prèvia: Usuari creat a la BBDD de l'aplicació.
- Cas d'èxit:
    1. L'usuari indica nom i contrasenya.
    2. L'aplicació valida el nom i contrasenya a la BBDD.
    3. El sistema retorna la informació i el rol de l'usuari.
- Cas alternatiu 1:
    2a. Usuari no existeix.
    3a. L'aplicació mostra un error indicant usuari incorrecte.
- Cas alternatiu 2:
    2b. Usuari existeix però contrasenya incorrecta.
    3b. L'aplicació mostra un error indicant contrasenya incorrecta.

Cas d'ús: RF2 – Sortir de l'aplicació
- Actors: Administrador, RRHH, Mànager i Treballador
- Descripció: L'usuari tanca la sessió per sortir de l'aplicació.
- Condició prèvia: Usuari identificat anteriorment.
- Cas d'èxit:
    1. L'usuari prem a tancar sessió.

Cas d'ús: RF3 – Gestionar usuaris
- Actors: Administrador
- Descripció: Afegir/editar usuari a l'aplicació.
- Condició prèvia: Usuari identificat amb el rol d'administrador.
- Cas d'èxit:
    1. L'administrador indica al formulari totes les dades de l'usuari requerides.
    2. El sistema valida que l'usuari no existeix a la BBDD.
    3. El sistema guarda les dades d'usuari a la BBDD.
- Cas alternatiu 1:
    2a. El sistema valida que l'usuari existeix a la BBDD.
    3a. El sistema guarda l'usuari amb les noves dades a la BBDD.

Cas d'ús: RF3 – Gestionar usuaris
- Actors: Administrador
- Descripció: Eliminar usuari de l'aplicació.
- Condició prèvia: Usuari identificat amb el rol administrador.
- Cas d'èxit:
    1. L'administrador indica l'usuari a eliminar del sistema.
    2. El sistema elimina de BBDD l'usuari indicat.

Cas d'ús: RF4 – Gestionar candidats
- Actors: RRHH
- Descripció: Afegir/editar candidat a l'aplicació.
- Condició prèvia: Estar identificat amb el rol de RRHH.
- Cas d'èxit:
    1. L'usuari RRHH indica les dades del candidat requerides.
    2. El sistema valida que l'usuari no existeix a la BBDD.
    3. El sistema guarda el candidat amb les dades indicades a la BBDD.

- Cas alternatiu 1:
    2a. El sistema valida que l'usuari existeix a la BBDD.
    3a. El sistema modificat l'usuari existent amb les noves dades.

Cas d'ús: RF4 – Gestionar candidats
- Actors: RRHH
- Descripció: Eliminar candidat a l'aplicació.
- Condició prèvia: Estar identificat com a usuari amb rol RRHH.
- Cas d'èxit:
    1. L'usuari RRHH indica el candidat a eliminar.
    2. El sistema elimina l'usuari de la BBDD de l'aplicació.

Cas d'ús: RF5 – Visualitzar posicions obertes per departament
- Actors: RRHH
- Descripció: L'usuari vol veure les posicions obertes en una llista per departament.
- Condició prèvia: Estar identificat com a usuari amb el rol RRHH
- Cas d'èxit:
    1. L'usuari indica el departament.
    2. El sistema valida que el departament existeix a la BBDD.
    3. El sistema retorna una llista de posicions obertes pel departament indicat.
- Cas alternatiu 1:
    2a. El sistema valida que el departament no existeix a la BBDD.
    3a. El sistema retorna un missatge informant que no existeix el departament indicat.

Cas d'ús: RF6 – Visualitzar absències
- Actors: RRHH
- Descripció: Visualitzar les absències que es troben actualment a l'empresa.
- Condició prèvia: Estar identificat com a usuari amb rol RRHH.
- Cas d'èxit:
    1. L'usuari RRHH indica que vol obtindré el nombre d'absències.
    2. El sistema retorna el nombre d'absències actuals.

Cas d'ús: RF7 – Contactar candidats
- Actors: RRHH
- Descripció: L'usuari RRHH vol contactar via correu electrònic amb un dels candidats.
- Condició prèvia: Estar identificat com a usuari amb rol de RRHH.
- Cas d'èxit:
    1. L'usuari RRHH indica el candidat amb qui vol contactar i indica el missatge.
    2. El sistema valida que el candidat existeix.
    3. El sistema envia el correu electrònic al candidat i bloqueja el contacte pel candidat.
- Cas alternatiu 1:
    2a. El sistema valida que el candidat existeix però el correu es incorrecte.
    3a. El sistema mostra un error que el correu electrònic no es correcte.

Cas d'ús: RF8 – Gestionar absències
- Actors: Mànager
- Descripció: L'usuari amb rol Mànager vol acceptar o denegar les sol·licituds d'absències trameses pels treballadors assignats a aquest.
- Condició prèvia: Usuari identificat amb el rol de Mànager.
- Cas d'èxit:
    1. El Mànager indica l'absència que vol acceptar o denegar.
    2. El sistema valida que l'absència existeix a BBDD.

3. El sistema guarda el nou estat de l'absència a BBDD.

Cas d'ús: RF9 – Sol·licitar absència
- Actors: RRHH, Mànager i Treballador
- Descripció: L'usuari vol sol·licitar una nova absència per a que el mànager li aprovi o denegui.
- Condició prèvia: Usuari identificat amb el rol de RRHH, Mànager o Treballador.
- Cas d'èxit:
    1. L'usuari indica al formulari les dades requerides per sol·licitar una absència.
    2. El sistema guarda les dades d'absència a la BBDD.

Cas d'ús: RF10 – Gestionar dades personals
- Actors: RRHH, Mànager i Treballador
- Descripció: L'usuari vol poder modificar o afegir dades personals.
- Condició prèvia: Usuari identificat qualsevol dels rols de RRHH, Mànager i Treballador.
- Cas d'èxit:
    1. L'usuari indica les dades personals.
    2. El sistema valida que l'usuari existeix a la BBDD.
    3. El sistema modifica les dades a la BBDD.

Cas d'ús: RF11 – Gestionar candidatures
- Actors: RRHH
- Descripció: Afegir/editar candidatura a l'aplicació.
- Condició prèvia: Estar identificat com a usuari amb rol RRHH.
- Cas d'èxit:
    1. L'usuari indica els camps requerits a més del candidat i la posició oberta de la candidatura.
    2. El sistema valida que no existeix la candidatura pel candidat a la BBDD.
    3. El sistema guarda la candidatura a la BBDD.
- Cas alternatiu 1:
    2a. El sistema valida que ja existeix una candidatura a la BBDD.
    3a. El sistema retorna un missatge d'error indicat que la candidatura ja existeix.

Cas d'ús: RF11 – Gestionar candidatura
- Actors: RRHH
- Descripció: Eliminar candidatura a l'aplicació.
- Condició prèvia: Estar identificat com a usuari amb rol RRHH.
- Cas d'èxit:
    1. L'usuari indica la candidatura a eliminar.
    2. El sistema valida que existeix la candidatura a la BBDD.
    3. El sistema elimina la candidatura a la BBDD.

Cas d'ús: RF12 – Visualitzar estat candidatura
- Actors: RRHH i Candidat
- Descripció: L'usuari vol visualitzar en quin estat es troba la candidatura.
- Condició prèvia: Estar identificat com a usuari amb rol RRHH o Candidat i que s'hagi creat una candidatura prèviament.
- Cas d'èxit:
    1. L'usuari indica que vol veure l'estat de la candidatura.
    2. El sistema valida que la candidatura existeix a la BBDD.
    3. El sistema retorna l'estat de la candidatura.
- Cas alternatiu 1:
    2a. El sistema valida que la candidatura no existeix a la BBDD.

3a. El sistema retorna un missatge informant que no existeix la candidatura a BBDD.

Cas d'ús: RF13 – Canviar estat candidatura

- Actors: RRHH
- Descripció: L'usuari vol actualitzar l'estat de la candidatura.
- Condició prèvia: Estar identificat prèviament com a usuari amb rol RRHH i que existeixi la candidatura a BBDD.
- Cas d'èxit:
    1. L'usuari indica la candidatura la qual vol canviar el seu estat.
    2. El sistema valida que la candidatura existeix a la BBDD.
    3. El sistema guarda el nou estat de la candidatura a la BBDD.
- Cas alternatiu 1:
    2a. El sistema valida que la candidatura no existeix a la BBDD.
    3a. El sistema mostra un missatge informant que la candidatura no existeix a BBDD.

Cas d'ús: RF14 – Assignar posició oberta a departament

- Actors: RRHH
- Descripció: L'usuari vol assignar una posició oberta al departament que requereix un treballador nou.
- Condició prèvia: Estar identificat com a usuari amb rol RRHH i que existeixi un departament i posició oberta.
- Cas d'èxit:
    1. L'usuari indica la posició oberta i departament.
    2. El sistema valida que la posició oberta existeix a la BBDD.
    3. El sistema valida que existeix el departament a la BBDD.
    4. El sistema assigna la posició oberta al departament.
- Cas alternatiu 1:
    2a. El sistema valida que no existeix la posició oberta a la BBDD.
    4a. El sistema retorna un missatge d'error que no existeix la candidatura.
- Cas alternatiu 2:
    3b. El sistema valida que no existeix el departament a la BBDD.
    4b. El sistema retorna un missatge d'error que no existeix el departament.

Cas d'ús: RF15 – Gestionar departaments

- Actors: Administrador
- Descripció: Afegir/Editar departaments a l'aplicació.
- Condició prèvia: Estar identificat com a Administrador
- Cas d'èxit:
    1. L'usuari introdueix el nom del departament.
    2. El sistema valida que no existeix el departament a la BBDD.
    3. El sistema guarda el departament a la BBDD.
- Cas alternatiu 1:
    2a. El sistema valida que existeix el departament a la BBDD.
    3a. El sistema guarda el departament modificant les dades a la BBDD.

Cas d'ús: RF15 – Gestionar departaments

- Actors: Administrador
- Descripció: Eliminar departament
- Condició prèvia: Estar identificat com a Administrador.
- Cas d'èxit:
    1. L'usuari indica el departament a esborrar.
    2. El sistema valida que el departament existeix a la BBDD.
    3. El sistema elimina el departament de la BBDD.

- Cas alternatiu 1:
    2a. El sistema valida que el departament no existeix a la BBDD.
    3a. El sistema mostra un missatge informant que el departament no existeix a la BBDD.

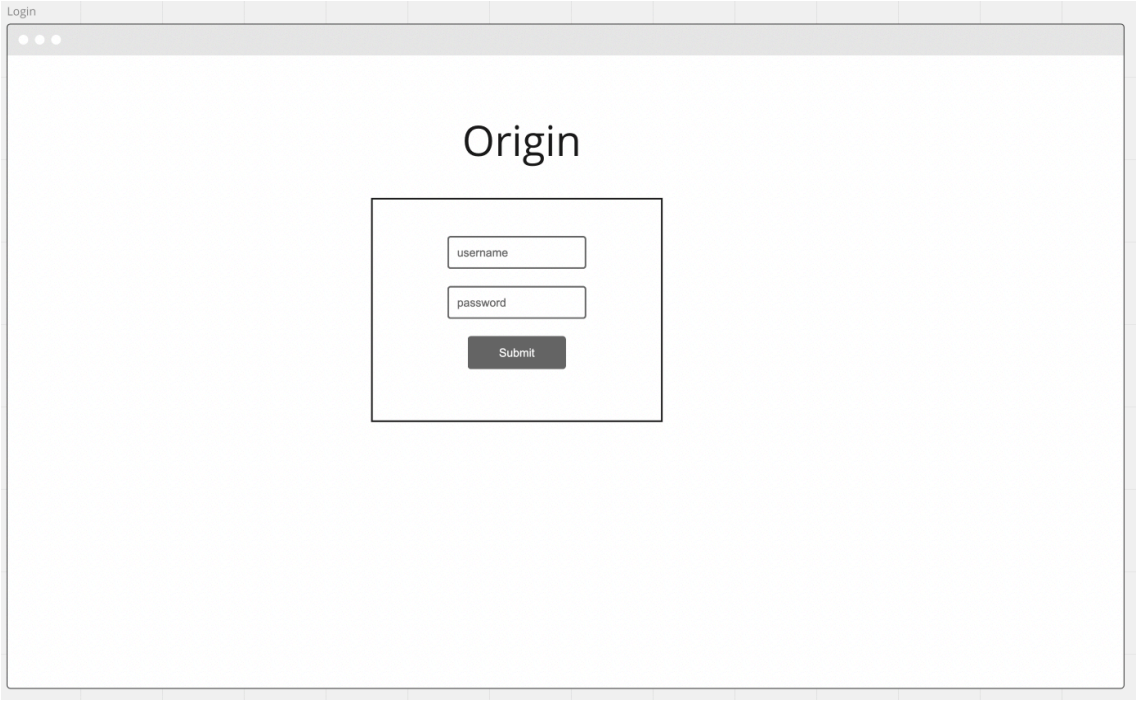
Cas d'ús: RF16 – Gestionar posicions obertes
- Actors: RRHH
- Descripció: Afegir/editar posicions obertes a l'aplicació.
- Condició prèvia: Estar identificat com a usuari amb rol RRHH.
- Cas d'èxit:
    1. L'usuari indica les dades requerides per a una posició oberta.
    2. El sistema valida que no existeix la posició oberta a la BBDD.
    3. El sistema guarda la posició oberta a la base de dades.
- Cas alternatiu 1:
    2a. El sistema valida que la posició oberta existeix a la BBDD.
    3a. El sistema guarda la posició oberta amb les noves dades a la BBDD.

Cas d'ús: RF16 – Gestionar posicions obertes
- Actors: RRHH
- Descripció: Eliminar posició oberta a l'aplicació.
- Condició prèvia: Estar identificat com a usuari amb rol RRHH.
- Cas d'èxit:
    1. L'usuari indica la posició oberta a esborrar.
    2. El sistema valida que la posició oberta existeix a la BBDD.
    3. El sistema esborra la posició oberta a la BBDD.

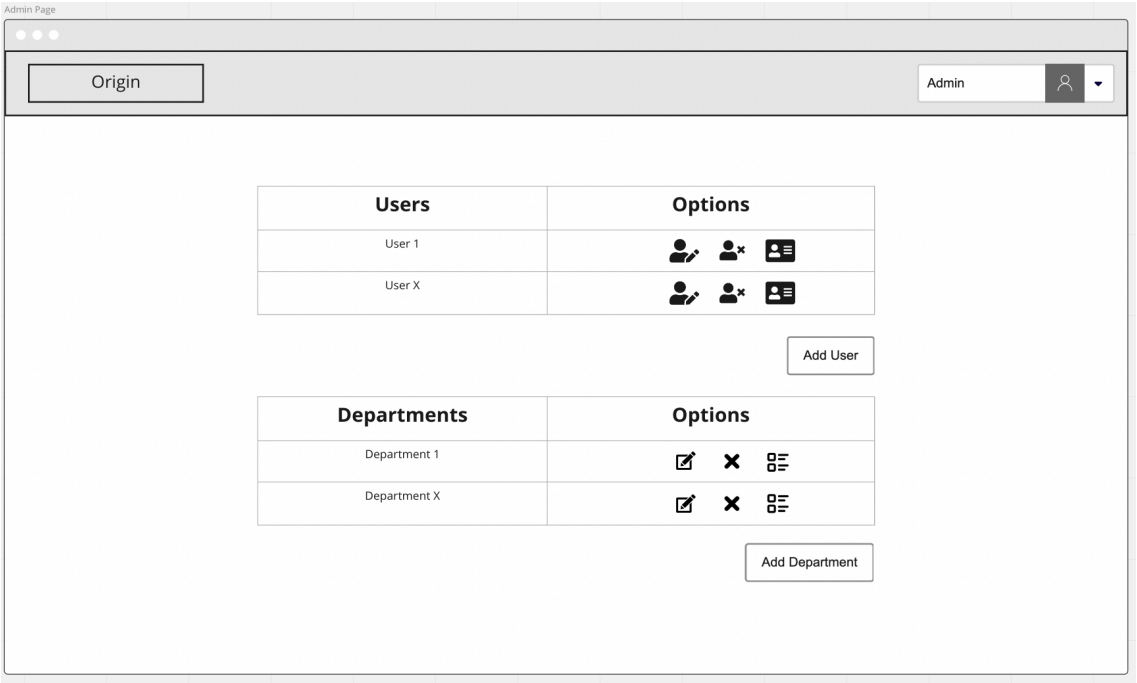
Cas d'ús: RF17 – Assignar candidat a posició oberta
- Actors: RRHH
- Descripció: L'usuari vol assignar un candidat a una posició oberta per iniciar el procés de selecció.
- Condició prèvia: Estar identificat com a usuari amb rol RRHH i que existeixin la posició oberta i el candidat.
- Cas d'èxit:
    1. L'usuari indica la posició oberta i el candidat.
    2. El sistema valida que existeix la posició oberta a la BBDD.
    3. El sistema valida que existeix el candidat a la BBDD.
    4. El sistema assigna el candidat a la posició oberta a la BBDD.
- Cas alternatiu 1:
    2a. El sistema valida que no existeix la posició oberta a la BBDD.
    4a. El sistema mostra un missatge informant que no existeix la posició oberta a la BBDD.
- Cas alternatiu 2:
    2b. El sistema valida que no existeix el candidat a la BBDD.
    4b. El sistema mostra un missatge informant que no existeix el candidat a la BBDD.

3.5. Model de pantalles – Prototipus

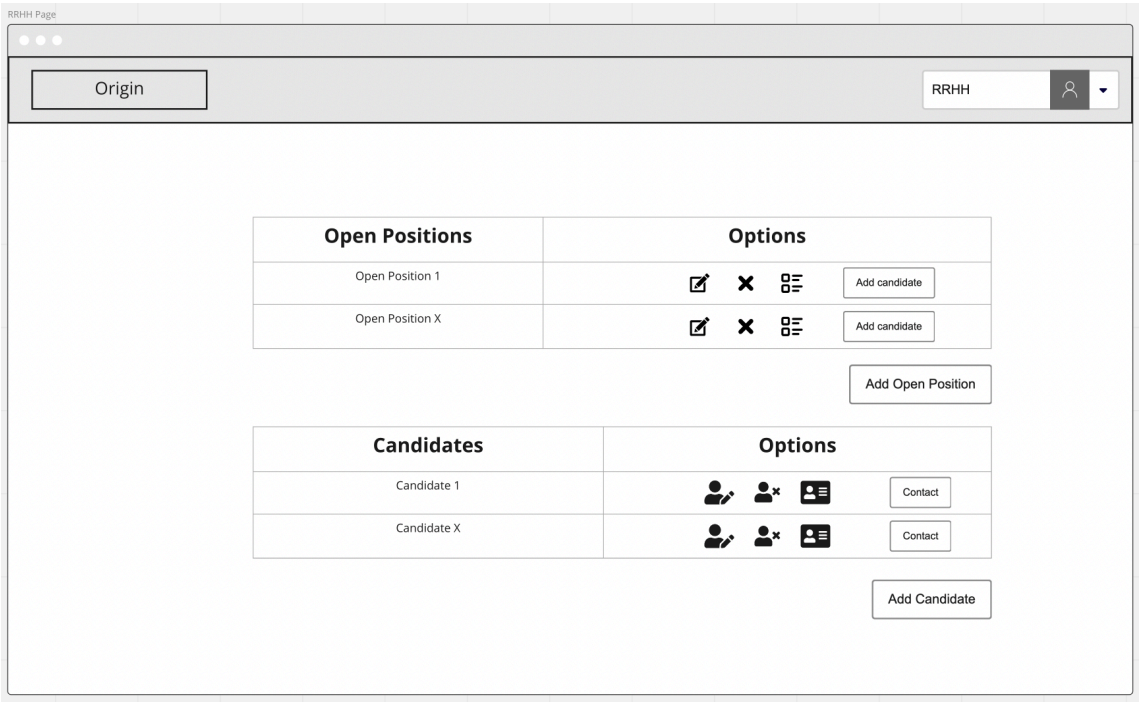
En aquest apartat s'obtenen les diferents interfícies de l'aplicació per obtenir una primera idea de com serà visualment la plataforma de gestió de recursos humans:



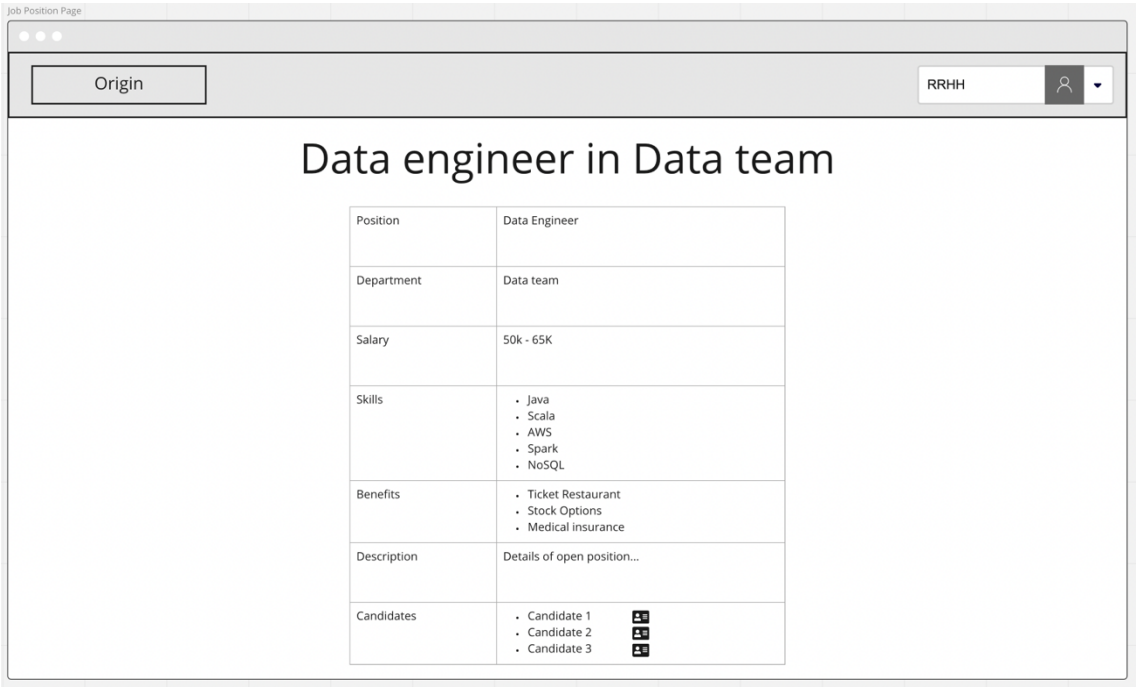
Il·lustració 7: Pantalla Login



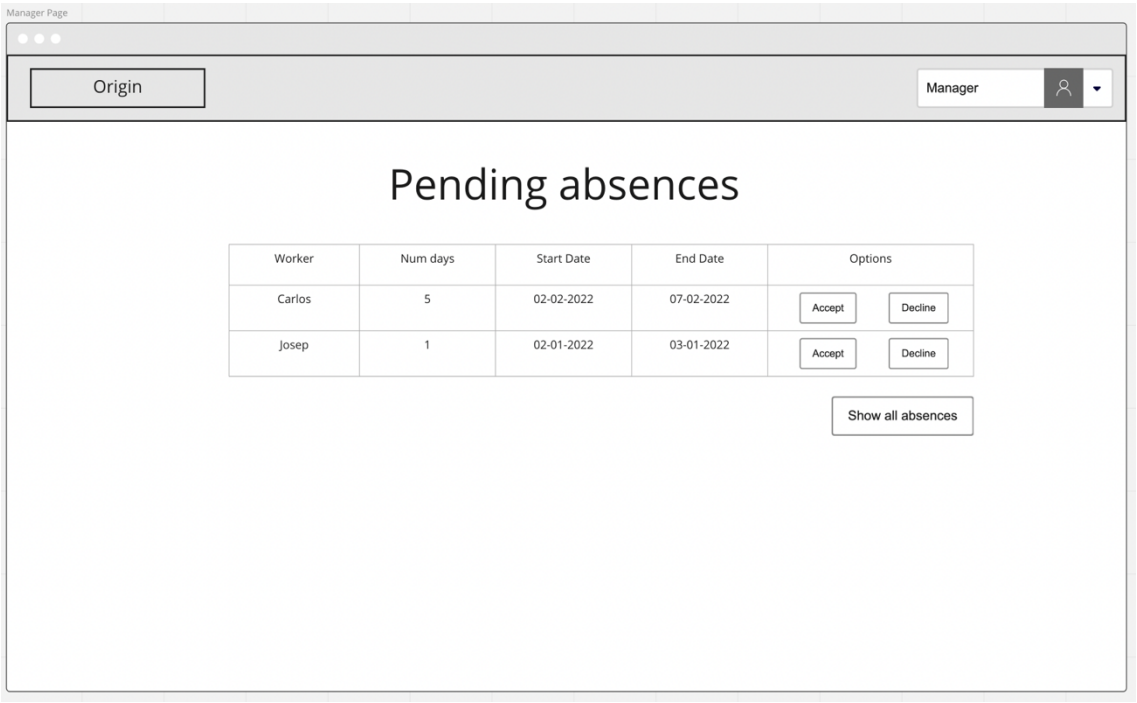
Il·lustració 8: Pantalla Administrador



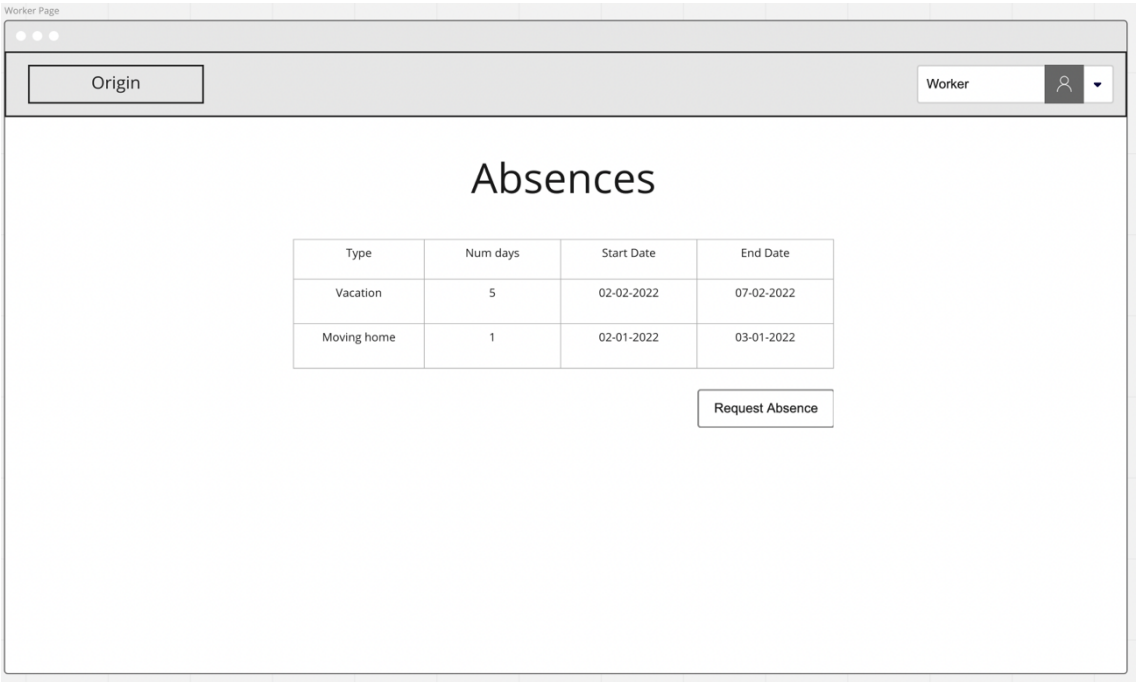
Il·lustració 9: Pantalla RRHH



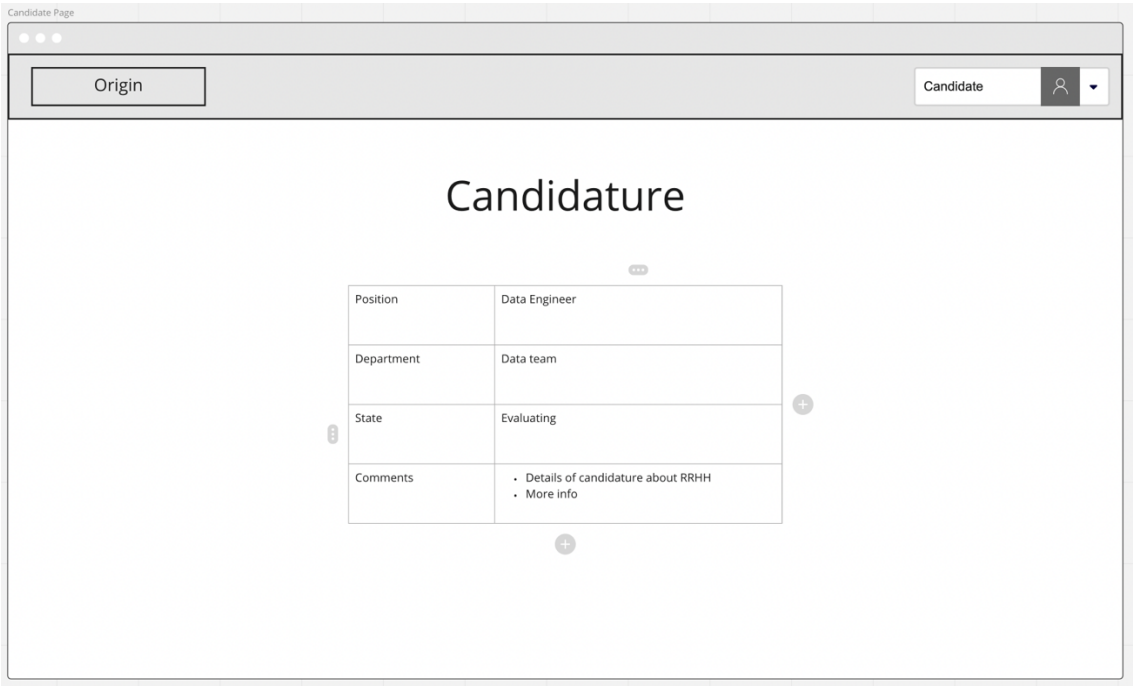
Il·lustració 10: Pantalla Open Position



Il·lustració 11: Pantalla manager



Il·lustració 12: Pantalla Worker



Il·lustració 13: Pantalla Candidate

3.6. Disseny de l'arquitectura

Per l'aplicació de gestió de recursos humans l'arquitectura que s'adapta millor és l'arquitectura heterogènia on trobem un client-servidor que s'organitzaran en capes. Però cada component de les capes o nivells l'estructurarem en una arquitectura orientada a objectes distribuïts.

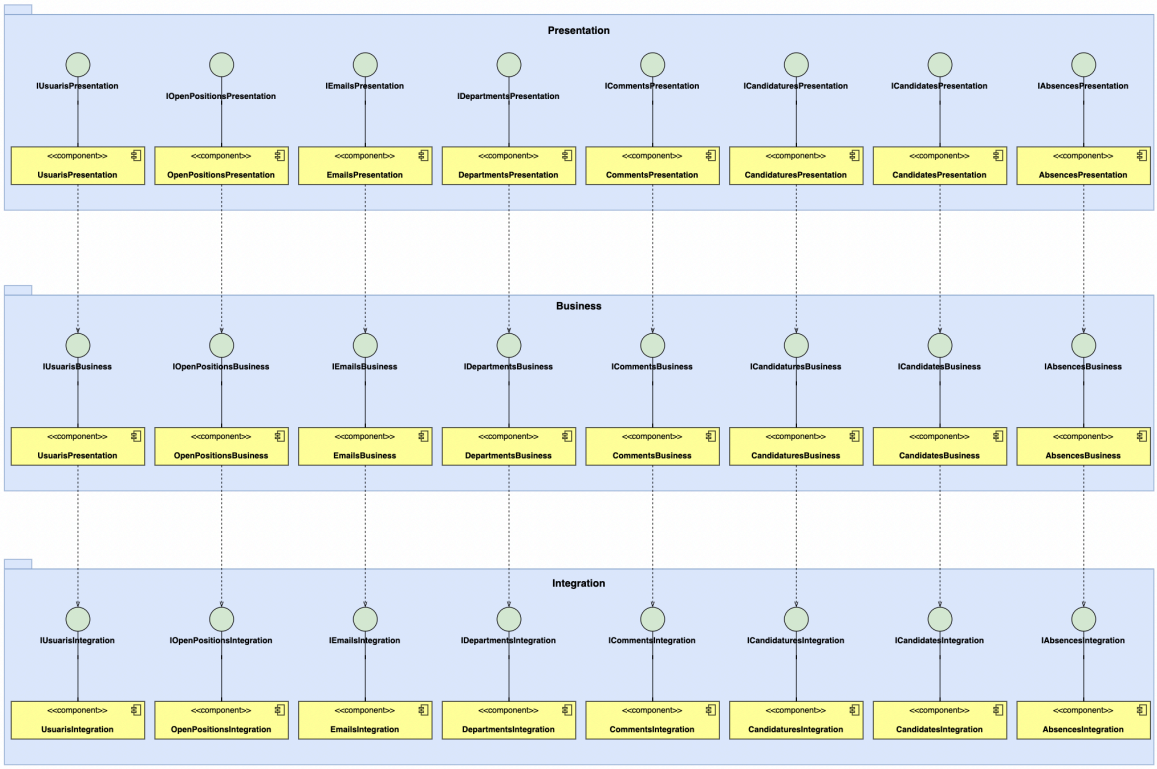
D'aquesta manera podem tenir un sistema amb un disseny basat en nivells d'abstracció clarament definits on es podran realitzar optimitzacions i refinaments de manera natural i, per tant, el sistema estarà obert a una àmplia reutilització. En resum, aconseguirem un sistema amb tolerància a fallades, fàcilment mantenible, flexible, escalable i poc acoblat com hem explicat anteriorment.

Mitjançant l'arquitectura per capes o nivells arribem a un aïllament de les funcionalitats a través d'interfícies ben definides on els usuaris puguin interactuar amb cadascun dels diferents serveis que ofereix el sistema.

Finalment utilitzant l'arquitectura orientada a objectes distribuïts el que volem precisament es que cadascuna de les capes del sistema sigui més escalable a i flexible on podem l'estructura estarà molt oberta a afegir noves màquines si escau.

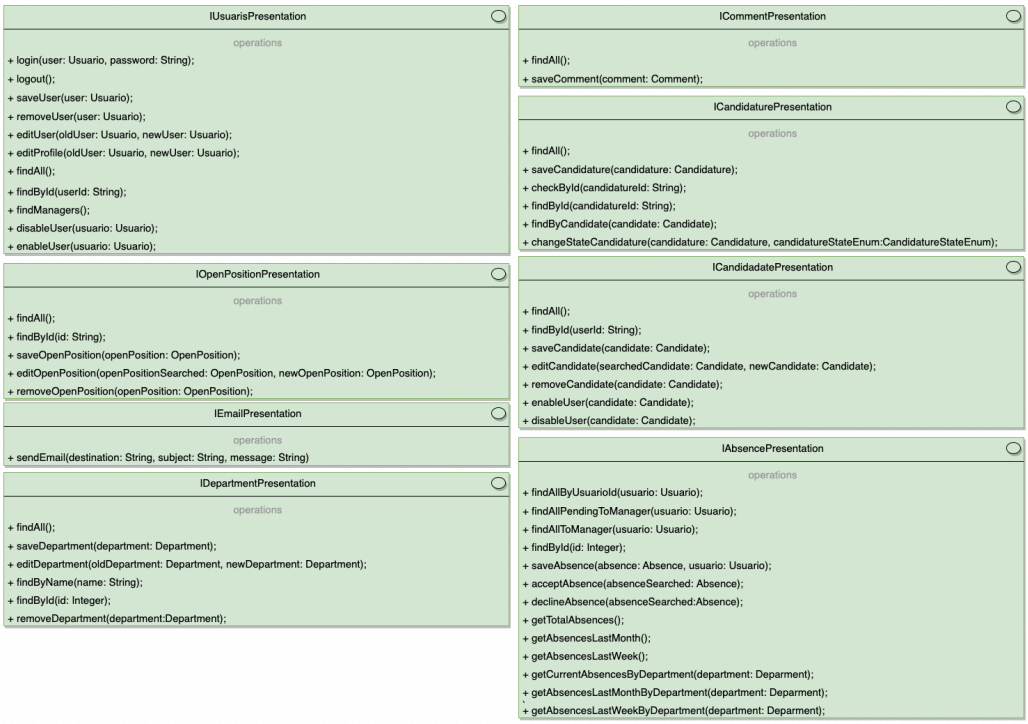
3.7. Diagrama de components

D'aquesta manera podem observar com el nostre sistema l'implementem amb 3 capes on una primera seria el client per la presentació, un altre per servidor on es processarà l'aplicació i finalment l'última on tenim el servidor de base de dades on es realitzaran les consultes de les dades.



Il·lustració 14: Diagrama de components

En aquest cas solament mostrarem la capa de presentació dels serveis dels components per evitar redundància, ja que són pràcticament amb totes les funcions que s'implementaran posteriorment:



Il·lustració 15: Serveis dels components

4. Implementació de l'aplicació

4.1. Configuració de l'entorn

En aquest apartat s'explicarà com s'ha de configurar l'entorn per a poder treballar com a desenvolupador de l'aplicació així com a poder realitzar proves en local.

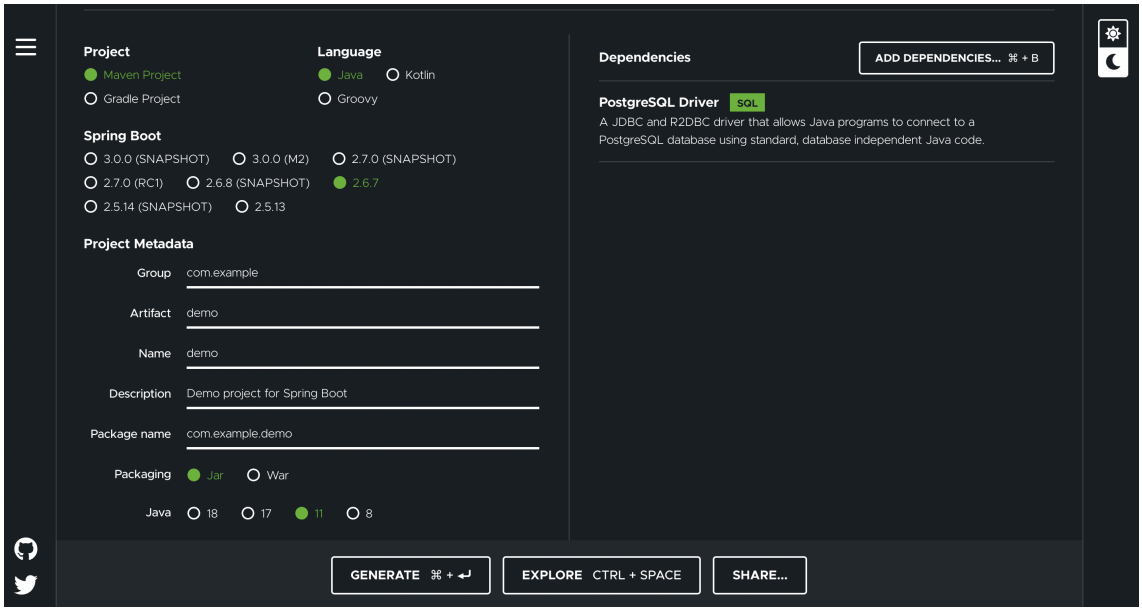
Primer de tot al final de la memòria es troba un annex amb la instal·lació de tot el programari necessari per inicialitzar l'aplicació en local.

L'entorn s'ha configurat per a desenvolupar l'aplicació en local mitjançant la instal·lació de la versió de Java 11, Maven amb la versió 3.6.2 i PostgreSQL versió 12.10.

Un cop dutes a terme les instal·lacions pertinents definides a l'annex comencem amb el *framework* utilitzat per desenvolupar l'aplicació web anomenant Spring Boot. Aquest *framework* ens permet crear aplicacions oblidant-nos de l'arquitectura, ja que són autocontigudes i, per tant, ens enfoquem directament en el desenvolupament del codi. La part de configuració de l'aplicació així com les dependències són gestionades pel *framework*.

Spring Boot ens proporciona una eina anomenada Initializr que permet generar un projecte Java web seleccionant la versió de Java amb la qual volem desenvolupar l'aplicació així com les llibreries que necessitarem per a una versió inicial del projecte com seria el connector de PostgreSQL per connectar-nos a la base de dades configurada a l'annex.

A la següent imatge es mostra una captura de Spring Initializr on seleccionem els paràmetres per generar el nostre projecte Maven:



Il·lustració 16: Spring Initializr

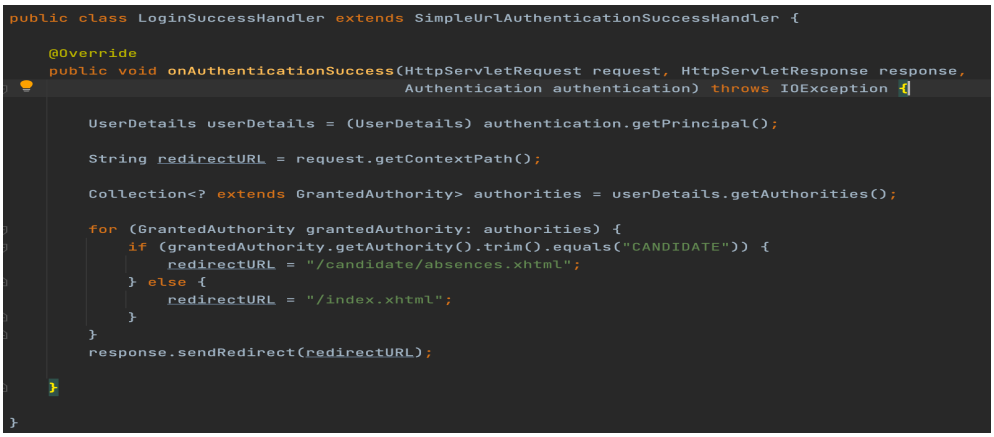
Un cop inserits els valors necessaris llavors la web et genera un fitxer comprimit amb extensió ".zip" que es podrà extreure i importar a l'Ide IntelliJ Idea per a començar a desenvolupar l'aplicació.

Els fitxers més importants d'un projecte utilitzant Maven i Spring Boot serien:

1. Pom.xml: Fitxer en format XML dedicat exclusivament a la configuració pròpia del projecte on es defineixen valors com el nom del projecte, les dependències necessàries o el *plugin* per a l'empaquetament del projecte.

2. Application.properties: Fitxer dedicat a les configuracions de l'aplicació com per exemple poden ser les credencials de la base de dades o les credencials per enviar correus electrònics per mitjà de Gmail així com el port amb el qual s'iniciarà l'aplicació per accedir de manera local mitjançant el navegador.

   Exemple d'algunes configuracions de l'aplicació:



```
public class LoginSuccessHandler extends SimpleUrlAuthenticationSuccessHandler {

    @Override
    public void onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse response,
                                        Authentication authentication) throws IOException {

        UserDetails userDetails = (UserDetails) authentication.getPrincipal();

        String redirectURL = request.getContextPath();

        Collection<? extends GrantedAuthority> authorities = userDetails.getAuthorities();

        for (GrantedAuthority grantedAuthority: authorities) {
            if (grantedAuthority.getAuthority().trim().equals("CANDIDATE")) {
                redirectURL = "/candidate/absences.xhtml";
            } else {
                redirectURL = "/index.xhtml";
            }
        }
        response.sendRedirect(redirectURL);

    }
}
```

Il·lustració 17: Exemple application.properties

Si es desitja importar i executar l'aplicació en local hem de seguir els següents passos:

1. Desempaquetar el projecte i accedir pel terminal a la carpeta del projecte.

2. Executar la següent comanda per instal·lar els temes de Primefaces:

   ```
   $ mvn install:install-file -Dfile=all-themes-1.0.10.jar -
   DgroupId=org.primefaces.themes -DartifactId=all-themes -
   Dversion=1.0.10 -Dpackaging=jar
   ```

3. Executar la següent comanda per compilar el projecte:

   ```
   $ mvn clean install
   ```

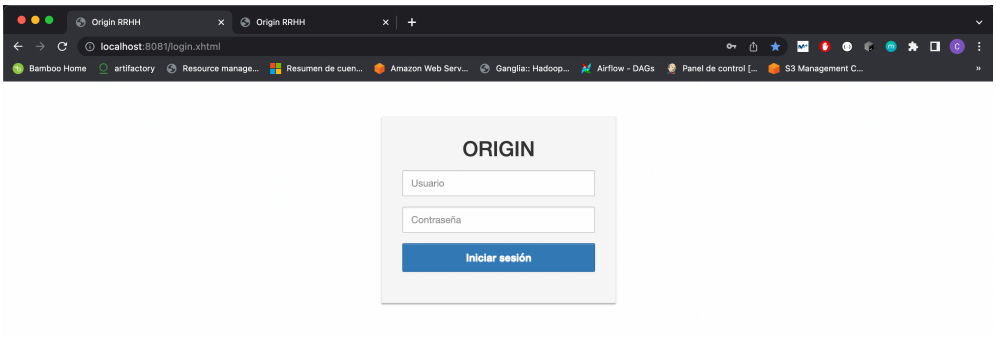
4. Executar la comanda per inicialitzar l'aplicació:

   ```
   $ mvn spring-boot:run
   ```

```java
public class LoginFailureHandler extends BaseBean implements AuthenticationFailureHandler{

    @Autowired
    private ServletContext context;

    @Override
    public void onAuthenticationFailure(
            HttpServletRequest request,
            HttpServletResponse response,
            AuthenticationException exception)
            throws IOException, ServletException {
            if (exception.getMessage().equals("Bad credentials")){
                request.setAttribute( s: "messageError",  o: "login.malas_credenciales");
            } else {
                request.setAttribute( s: "messageError",  o: "login.inesperado");
            }
        RequestDispatcher dispatcher = context.getRequestDispatcher( s: "/login.xhtml");
        dispatcher.forward(request, response);
    }

}
```

Il·lustració 18: Inicialització aplicació

5. Obrir el cercador d'internet i introduir la direcció http://localhost:8081/ per accedir a l'aplicació.



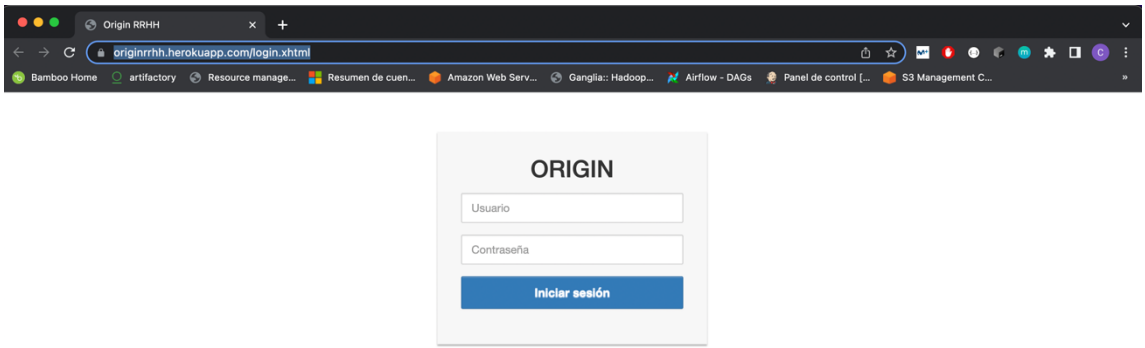
Il·lustració 19: Accés local

6. Si volem accedir com a administrador un cop arribem a la pàgina d'inici de sessió llavors utilitzarem les credencials per defecte de l'administrador que són:

- User: admin
- Pass: 1234

4.2. Accés online

Aquesta seria la forma per configurar, desenvolupar i inicialitzar l'aplicació, però també es troba l'alternativa d'utilitzar l'aplicació directament, ja que aquesta ha estat publicada directament a un servidor públic d'Heroku amb domini propi a la direcció https://originrrhh.herokuapp.com/login.xhtml



Il·lustració 20: Accés Heroku

4.3. Estructura del projecte

El projecte està basat en una estructura de paquets amb 6 paquets a més de les classes pròpies de Spring per a configurar algunes propietats com la seguretat de l'aplicació o la classe principal que ha de ser-hi a primer nivell.

La resta dels paquets són els següents:

- **Beans:** Objectes que seran instanciats i administrats pel contenidor de Spring. Aquests objectes seran els encarregats d'enviar la informació a la vista.

```java
@Component
@Scope("view")
public class ProfileBean extends BaseBean implements Serializable {

    private final IUsuarioService usuarioService;

    private Usuario usuari = new Usuario();
    private List<IdiomaEnum> idiomas = new ArrayList<>(Arrays.asList(IdiomaEnum.values()));

    public ProfileBean(IUsuarioService usuarioService) { this.usuarioService = usuarioService; }

    @PostConstruct
    public void init() { loadDetails(); }

    private void loadDetails() { usuari = usuarioService.findById(getUsuario().getUsuari()); }

    public Usuario getUsuari() { return usuari; }

    public void setUsuari(Usuario usuari) { this.usuari = usuari; }

    public void editProfile() {
        Usuario searchedUser = usuarioService.findById(usuari.getUsuari());
        usuarioService.editProfile(searchedUser, usuari);
        usuari = new Usuario();
        loadDetails();
        showInfo(getMessage( clave: "info.usuario_edit"));
    }

    public List<IdiomaEnum> getIdiomas() { return idiomas; }

    public void setIdiomas(List<IdiomaEnum> idiomas) { this.idiomas = idiomas; }
}
```

Il·lustració 21: Bean Profile

- **Converter:** Classes d'utilitat per objectes propis a l'hora d'enviar objectes des del *frontEnd* cap al *backend* mitjançant formularis.

```java
@FacesConverter("departmentConverter")
@Component("departmentConverter")
public class DepartmentConverter implements Converter {

    @Autowired
    IDepartmentService departmentService;

    @Override
    public Object getAsObject(FacesContext context, UIComponent component, String value) {
        if (value == null || value.length() == 0) {
            return null;
        }
        return departmentService.findById(Integer.parseInt(value));
    }

    @Override
    public String getAsString(FacesContext context, UIComponent component, Object value) {
        if (value == null || value.equals("")) {
            return null;
        }
        return ((Department)value).getId().toString();
    }

}
```

Il·lustració 22: Converter Department

- **Enums:** Classes per definir enumeracions pròpies dels atributs dels models.

```
(* |mediabuying-sbx:fargate) cliarte@LFVFF72PNQ05N ~ cd Desktop/Ingenieria\ Informática/TFG/TFG\ Carlos/origin
(* |mediabuying-sbx:fargate) cliarte@LFVFF72PNQ05N ~/Desktop/Ingenieria Informática/TFG/TFG Carlos/origin  main mvn install:install-file -Dfile=all-themes-1.0.10.jar -DgroupId=org.primefaces.themes -DartifactId=all-themes -Dversion=1.0.10 -Dpackaging=jar
[WARNING]
[WARNING] Some problems were encountered while building the effective settings
[WARNING] 'profiles.profile[artifactory].pluginRepositories.pluginRepository.id' must not be 'local', this identifier is reserved for the local repository, using it for other repositories will corrupt your repository metadata. @ /Users/cliarte/.m2/settings.xml
[WARNING]
[INFO] Scanning for projects...
[WARNING]
[WARNING] Some problems were encountered while building the effective model for com.uoc:origin:jar:0.0.1-SNAPSHOT
[WARNING] 'pluginRepositories.pluginRepository.id' must not be 'local', this identifier is reserved for the local repository, using it for other repositories will corrupt your repository metadata.
[WARNING]
[WARNING] It is highly recommended to fix these problems because they threaten the stability of your build.
[WARNING]
[WARNING] For this reason, future Maven versions might no longer support building such malformed projects.
[WARNING]
[INFO]
[INFO] ------------------------< com.uoc:origin >------------------------
[INFO] Building origin 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-install-plugin:2.5.2:install-file (default-cli) @ origin ---
[INFO] Installing /Users/cliarte/Desktop/Ingenieria Informática/TFG/TFG Carlos/origin/all-themes-1.0.10.jar to /Users/cliarte/.m2/repository/org/primefaces/themes/all-themes/1.0.10/all-themes-1.0.10.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  1.115 s
[INFO] Finished at: 2022-05-09T17:44:18+02:00
[INFO] ------------------------------------------------------------------------
(* |mediabuying-sbx:fargate) cliarte@LFVFF72PNQ05N ~/Desktop/Ingenieria Informática/TFG/TFG Carlos/origin  main
```

Il·lustració 23: Enum Rols

- **Model:** Classes que defineixen les entitats i els atributs d'aquestes dins l'aplicació.

```java
@Entity
public class Candidate extends Usuario implements Serializable {

    @OneToOne(mappedBy = "candidate")
    private Candidature candidature;

    public Candidature getCandidature() { return candidature; }

    public void setCandidature(Candidature candidature) { this.candidature = candidature; }
}
```

Il·lustració 24: Model Candidate

- **Repository:** Classes per realitzar consultes a la base de dades.

```java
public interface IAbsenceRepo extends JpaRepository<Absence, String> {

    @Query("SELECT a FROM Absence a WHERE a.worker = :usuario ORDER BY a.startDate DESC")
    List<Absence> findAllByUsuarioId(@Param("usuario") Usuario usuario);

    @Query("SELECT a FROM Absence a WHERE a.manager = :usuario AND a.state = :state ORDER BY a.startDate DESC")
    List<Absence> findAllPendingToManager(@Param("usuario") Usuario usuario, @Param("state") AbsenceStateEnum state);

    @Query("SELECT a FROM Absence a WHERE a.manager = :usuario ORDER BY a.startDate DESC")
    List<Absence> findAllToManager(@Param("usuario") Usuario usuario);

    Absence findById(Integer id);
}
```

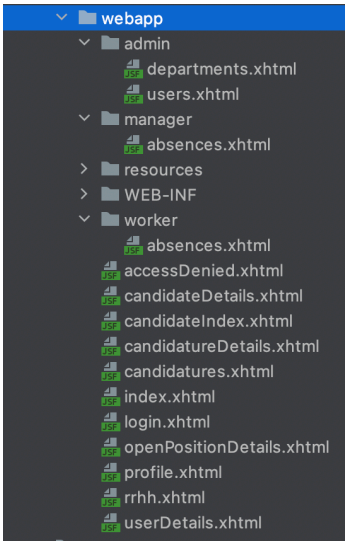
Il·lustració 25: Repositori Absence

- **Service:** Conté les classes de servei on es troba normalment la lògica de negoci i fan de nexe entre el repositori i les vistes.

```java
@Service
public class DepartmentServiceImpl implements IDepartmentService {

    private final IDepartmentRepo departmentRepo;

    public DepartmentServiceImpl(@Lazy IDepartmentRepo departmentRepo) { this.departmentRepo = departmentRepo; }

    @Override
    public List<Department> findAll() { return departmentRepo.findAll(); }

    @Override
    public Department findById(Integer id) { return departmentRepo.findById(id); }

    @Override
    public void saveDepartment(Department department) { departmentRepo.save(department); }

    @Override
    public Department findByName(String name) { return departmentRepo.findByName(name); }

    @Override
    public void editDepartment(Department oldDepartment, Department newDepartment) {
        oldDepartment.setName(newDepartment.getName());
        departmentRepo.save(oldDepartment);
    }
```

Il·lustració 26: Servei Department

A més després trobarem tota l'estructura de directoris del frontal de la mateixa aplicació on tenim les vistes a primer nivell i les vistes en funció del rol per a sincronitzar-se amb la seguretat de l'aplicació.



Il·lustració 27: Directori vistes

D'altra banda, trobem el directori *resources* on tenim la internacionalització de l'aplicació amb els següents fitxers per als dos idiomes seleccionats:

- Messages_en.properties: Fitxer que conté les traduccions de l'aplicació en anglès.

```
# HEADER
menu.title=ORIGIN
header.salir=Logout
header.perfil=Profile

# GLOBAL
global.confirmar=Confirm
global.borrar=Delete
global.acceso_denegado=Access denied
global.opciones=Options

# INDEX
index.title=Index
index.usuarios=Users
index.departamentos=Departments
index.posiciones_abiertas_candidatos=Open positions and candidates
index.candidaturas=Candidatures
index.candidate_candidaturas=Show my candidatures
index.manager_ausencias=Show Pending absences
index.trabajador_ausencias=Show my absences
```

Il·lustració 28: Fitxer locale en

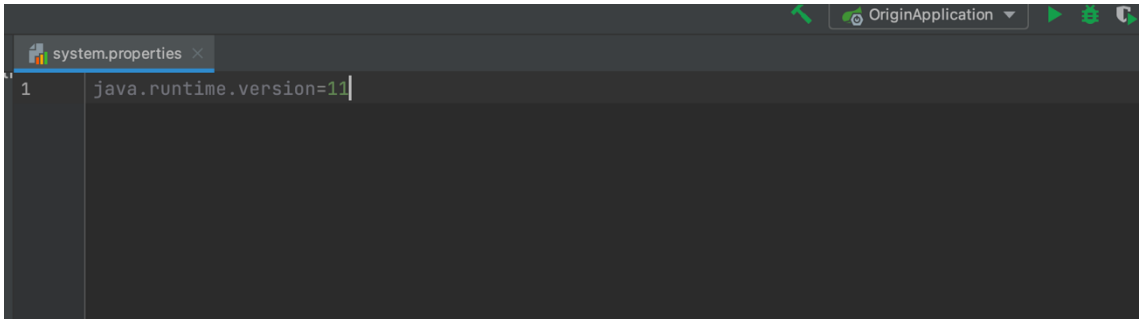
- Messages_es.properties: Fitxer que conté les traduccions de l'aplicació en castellà.

```
# HEADER
menu.title=ORIGIN
header.salir=Salir
header.perfil=Perfil

# GLOBAL
global.confirmar=Confirmar
global.borrar=Eliminar
global.acceso_denegado=Acceso denegado
global.opciones=Opciones

# INDEX
index.title=Índice
index.usuarios=Usuarios
index.departamentos=Departamentos
index.posiciones_abiertas_candidatos=Posiciones abiertas y candidatos
index.candidaturas=Candidaturas
index.candidate_candidaturas=Mostrar mis candidaturas
index.manager_ausencias=Mostrar ausencias pendientes
index.trabajador_ausencias=Mostrar mis ausencias
```

Il·lustració 29: Fitxer locale es

D'aquesta manera si es volgués afegir un altre idioma, seria tan fàcil com crear un nou fitxer amb la mateixa nomenclatura que la resta i traduir els textos a l'idioma desitjat a més d'afegir l'idioma a l'enumeració corresponent perquè l'aplicació el detectés.

Finalment, a l'arrel del projecte trobem un fitxer anomenat *system.properties* que serveix per definir opcions de la JVM de Java per arrencar l'aplicació amb Heroku, ja que per defecte la plataforma utilitza JDK 8 i el projecte s'ha desenvolupat amb JDK 11.



Il·lustració 30: Fitxer System Heroku

4.4. Mapa de l'aplicació

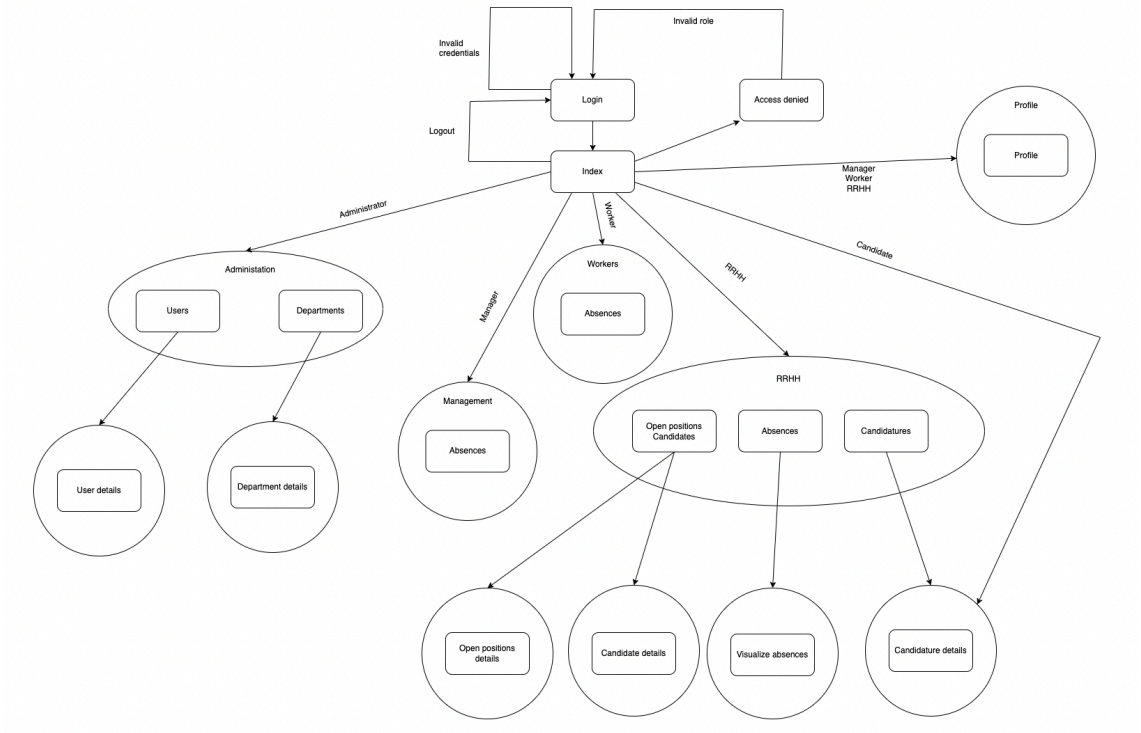
En aquest apartat trobem el mapa de l'aplicació en funció de les vistes per rol.

Tots els usuaris accedeixen a la plataforma via l'inici de sessió i aquesta redirigeix cap a dos llocs basant-se en l'autenticació de les credencials. La primera opció és una redirecció a la pàgina d'inici de sessió amb un missatge de credencials incorrectes.

D'altra banda, si les credencials són correctes llavors, l'usuari accedirà a la plataforma i serà redirigit a la pàgina índex on visualitzarà opcions basades al seu rol.

Un cop dintre la plataforma si un usuari vol accedir a una pàgina incorrecta a través de l'URL, llavors l'aplicació l'enviarà a la pàgina d'accés prohibit.

Tal com es mostra a la següent imatge es pot observar a quines pàgines podrà accedir l'usuari un cop accedeixi a la plataforma.



Il·lustració 31: Mapa aplicació

4.5. Capa de persistència

La capa de persistència és aquella que s'encarrega de gestionar la comunicació entre la capa de negoci i les base de dades de manera que persisteix els objectes mitjançant sentències SQL.

S'ha utilitzat Hibernate com a eina de mapeig objecte-relacional, ja que és relativament senzill incorporar-la a Spring Boot mitjançant l'ús de la dependència spring-boot-starter-data-jpa i el connector de PostgreSQL.

La dependència spring-boot-starter-data-jpa es fa servir bàsicament per a etiquetar les entitats o models del codi i d'aquesta manera poder crear fàcilment les entitats a la base de dades així com les dependències entre taules.

D'altra banda, trobem el connector PostgreSQL que bàsicament serveix per establir connexió amb les credencials pertinents contra la base de dades.

Tant una dependència com l'altra són essencials per a poder persistir els objectes de l'aplicació a la base de dades utilitzant sentències SQL com poden ser per exemple insert, *update* o *delete* entre altres.

A continuació es mostren exemples reals del codi i s'anirà explicant detalladament cadascuna de les parts que necessita la capa de persistència per funcionar correctament.

Exemple del model Departament:

```java
@Entity
public class Department implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(updatable = false, nullable = false)
    private Integer id;

    @Column(length = 30)
    private String name;

    @OneToMany(mappedBy = "department")
    private List<OpenPosition> openPositions = new ArrayList<>();

    @OneToMany(mappedBy = "department")
    private List<Usuario> users = new ArrayList<>();
```

Il·lustració 32: Model Department

Primer de tot mencionar la primera etiqueta nomenada @Entity que trobem a l'inici del model i que aquesta serveix per a definir aquest com a taula dintre de la base de dades. Per tant, un cop executem l'aplicació, es genera una taula Department si aquesta no existeix amb els camps corresponents.

A continuació es defineixen les etiquetes pròpies dels atributs del model o columnes a la base de dades. En aquest cas podem observar com el camp id conté tres etiquetes:

- **@Id:** Defineix que el camp que contingui aquesta etiqueta serà la clau primària de l'entitat.

- **@GeneratedValue:** Significa que el camp serà auto generat de manera automàtica i, a més, utilitzarà una estratègia de columna identitat a base de dades.

- **@Column:** Anotació per indicar que un atribut del model serà una columna de l'entitat a base de dades i, a més es poden definir atributs propis de columna com per exemple que sigui editable o nul. En aquest cas no es compleix la condició, però si és un camp de tipus text llavors també es pot indicar la mesura màxima del text inserit a la columna.

També podem trobar altres anotacions associades als atributs de l'entitat com els següents:

- **@Enumerated:** Defineix que l'atribut serà una enumeració definida al codi de l'aplicació.

- **@Transient:** Anotació per indicar que l'atribut no serà una columna de la taula final i que, per tant, el seu contingut es calcularà a partir d'altres camps de la taula.

Podem observar aquestes últimes anotacions on trobem l'atribut reason definit com a una enumeració i el numDays com a Transient, ja que serà un càlcul de la resta de la data final menys la inicial a la definició de l'entitat d'absències:

```java
@Entity
public class Absence implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(updatable = false, nullable = false)
    private Integer id;

    @Enumerated(EnumType.STRING)
    private AbsenceReasonEnum reason;

    @Column
    private Date startDate;

    @Column
    private Date endDate;

    @Transient
    private Integer numDays;

    @Enumerated(EnumType.STRING)
    private AbsenceStateEnum state;
```

Il·lustració 33: Model Absence

Aquesta dependència també ens permet mitjançant anotacions les relacions entre les entitats, ja que utilitzem una base de dades relacional i, per tant, les taules tindran claus foranes per interrelacionar-se. Les anotacions referents a aquests tipus de relacions són les següents:

- **@OneToMany:** Defineix que l'atribut anotat tindrà una relació 1-n respecte de l'entitat destí.

- **@ManytoOne:** Defineix que l'atribut anotat tindrà una relació n-1 respecte de l'entitat destí.

- **@OneToOne:** Defineix que l'atribut anotat tindrà una relació 1-1 respecte de l'entitat destí.

- **@ManyToMany:** Defineix que l'atribut anotat tindrà una relació n-n respecte l'entitat destí.

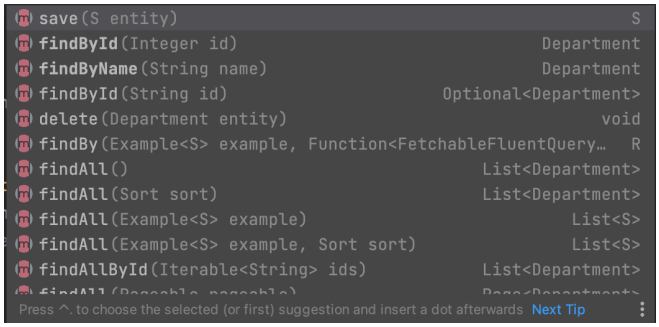
Finalment, un cop definit el model i definides les anotacions pertinents llavors s'ha de desenvolupar el repositori per a realitzar les consultes contra la base de dades. Per tant, si observem l'exemple de l'entitat Department, trobem el respectiu repositori:

```
@Repository
public interface IDepartmentRepo extends JpaRepository<Department, String> {
    Department findById(Integer id);

    Department findByName(String name);
}
```

Il·lustració 34: Repositori Department

En aquest cas utilitzant la dependència jpa nomenada anteriorment solament s'ha d'anotar la interfície amb l'anotació @Repository perquè Spring la detecti com a repositori i definir el tipus de dades que retornaran les sentències SQL.

Tal com es pot observar la interfície estén de la JpaRepository que per defecte conté funcions per a realitzar algunes consultes bàsiques en funció dels atributs com per exemple:



Il·lustració 35: Llista funcions repositori

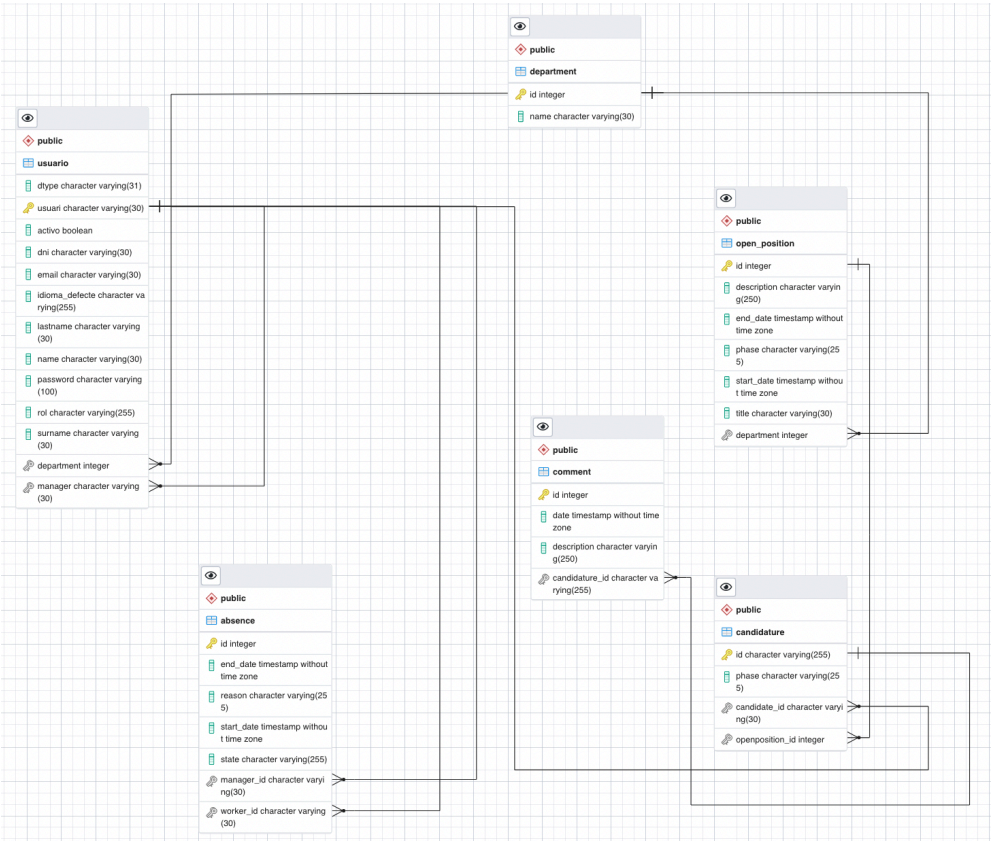
Però si volem realitzar consultes personalitzades en funció de les dades que volem extreure de la base de dades llavors s'utilitza l'anotació @Query i a més es pot definir paràmetres perquè la consulta sigui en funció de valors passats com a paràmetres a la funció de la següent manera:

```
public interface ICommentRepo extends JpaRepository<Comment, String> {
    @Query("SELECT c FROM Comment c WHERE c.candidature.id = :id")
    List<Comment> findByCandidature(@Param("id") String id);
}
```

Il·lustració 36: Repositori amb *query* pròpia

A l'exemple anterior es pot observar com passem l'identificador de la candidatura per a cercar tots els comentaris associats a aquesta.

A continuació es mostra l'esquema generat a partir de la definició dels models que correspon al diagrama UML desenvolupat al disseny de l'aplicació:



Il·lustració 37: ERD base de dades

4.6. Capa de negoci

La capa de negoci és l'encarregada de definir tota la lògica de negoci de l'aplicació. Per tant, aquí trobarem totes les funcionalitats definides als diferents serveis nomenats anteriorment. Aquesta capa es troba entre les capes de presentació i persistència i, ofereix les dades obtingudes de la capa de persistència cap a les diferents vistes en funció dels *beans* utilitzats per les diferents pantalles.

A continuació es mostraran alguns exemples reals de codi de l'aplicació per a tenir més context de la capa de negoci.

El primer de tot que es necessita és crear una interfície per cada servei on es defineixen les diferents funcions:

```java
public interface IDepartmentService {
    List<Department> findAll();

    void saveDepartment(Department department);

    Department findByName(String name);

    void editDepartment(Department oldDepartment, Department newDepartment);

    Department findById(Integer id);

    void removeDepartment(Department department);
}
```

Il·lustració 38: Interfície Department

Com es pot observar a l'anterior imatge, les funcions solament estan definides sense implementar la lògica de negoci. Un cop tenim creada la interfície llavors es genera una altra classe que serà la implementació del servei de la següent manera:

```java
@Service
public class DepartmentServiceImpl implements IDepartmentService {

    private final IDepartmentRepo departmentRepo;

    public DepartmentServiceImpl(@Lazy IDepartmentRepo departmentRepo) { this.departmentRepo = departmentRepo; }

    @Override
    public List<Department> findAll() { return departmentRepo.findAll(); }

    @Override
    public Department findById(Integer id) {
        return departmentRepo.findById(id);
    }

    @Override
    public void saveDepartment(Department department) {
        departmentRepo.save(department);
    }

    @Override
    public Department findByName(String name) { return departmentRepo.findByName(name); }

    @Override
    public void editDepartment(Department oldDepartment, Department newDepartment) {
        oldDepartment.setName(newDepartment.getName());
        departmentRepo.save(oldDepartment);
    }

    @Override
    public void removeDepartment(Department department) { departmentRepo.delete(department); }
}
```

Il·lustració 39: Implementació interfície Department

Primer de tot, s'utilitza l'anotació @Service perquè Spring detecti aquesta classe com a servei i, per tant, la carregui al context de l'aplicació. A partir d'aquí s'ha d'injectar el repositori pertinent, ja que aquesta classe cridarà funcions del repositori perquè la capa de persistència extregui dades de la base de dades mitjançant sentències SQL.

Per evitar problemes de dependències cícliques d'injecció de dependències es fa ús de l'anotació @Lazy i, per tant, aquesta dependència no es carregarà més d'una vegada al context de Spring.

En aquest cas les funcions que trobem implementades al servei no tenen molta lògica de negoci, però sí que es pot apreciar que la funció editDepartment està assignant un nou nom al nou objecte departament que es persistirà a base de dades per a realitzar una edició de manera satisfactòria.

A la capa de negoci també es troba la part de l'enviament de correus electrònics per utilitzar-se en diferents funcionalitats de l'aplicació per informar de l'activitat en tot moments als diferents actors implicats al cas d'ús pertinent.

A continuació es mostra el codi de la classe EmailServiceImpl:

```java
@Service
public class EmailServiceImpl implements IEmailService {

    private final JavaMailSender javaMailSender;

    public EmailServiceImpl(@Lazy JavaMailSender javaMailSender) { this.javaMailSender = javaMailSender; }

    @Override
    public void sendEmail(String destination, String subject, String message) throws MailException {
        MimeMessagePreparator preparator = mimeMessage -> {
            mimeMessage.setRecipient(Message.RecipientType.TO,
                    new InternetAddress(destination));
            mimeMessage.setFrom(new InternetAddress("originrrhh@gmail.com"));
            mimeMessage.setSubject(new String(subject.getBytes(StandardCharsets.ISO_8859_1), StandardCharsets.UTF_8),  charset: "UTF-8");
            mimeMessage.setText(new String(message.getBytes(StandardCharsets.UTF_8), StandardCharsets.UTF_8),  charset: "UTF-8");
        };
        try{
            javaMailSender.send(preparator);
        }
        catch (MailException ex) {
            System.err.println(ex.getMessage());
        }
    }

}
```

Il·lustració 40: Implementació interfície Email

A l'exemple anterior trobem com s'implementa una funcionalitat per enviar correus electrònics. En aquest cas s'ha fet servir la classe MimeMessage amb una preparació prèvia dels textos per a solucionar errors amb els caràcters especials i que els missatges tinguin una llegibilitat correcta. Per tant, els textos són transformats a UTF-8 per fer servir caràcters del castellà.

Finalment, es mostra un altre exemple de lògica de negoci situat al servei d'absències per extreure absències per franges de temps en funció del departament.

```java
    private Map<String, Date> calculateLastWeekDates(){
        Map<String, Date> dates = new HashMap<>();
        ZoneId defaultZoneId = ZoneId.systemDefault();
        Date firstDayLastWeek = Date.from(
                LocalDate.now() LocalDate
                        .minusWeeks(1)
                        .with(DayOfWeek.MONDAY)
                        .atStartOfDay(defaultZoneId) ZonedDateTime
                        .toInstant() Instant
        );
        Date lastDayLastWeek = Date.from(
                LocalDate.now() LocalDate
                        .minusWeeks(1)
                        .with(DayOfWeek.SUNDAY)
                        .atStartOfDay(defaultZoneId) ZonedDateTime
                        .toInstant() Instant
        );
        dates.put("firstDayLastWeek", firstDayLastWeek);
        dates.put("lastDayLastWeek", lastDayLastWeek);
        return dates;
    }
}
```

Il·lustració 41: Exemple lògica negoci 1

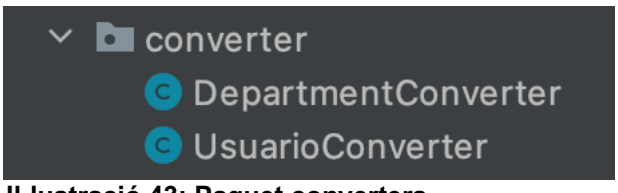
En aquest cas es defineix una funció privada al servei d'absències per a calcular el primer i últim dia de la setmana anterior per a retornar-los en un mapa clau valor.

Un cop calculats els dos dies llavors es recuperen els dos dies per a passar-los com a paràmetres juntament amb el departament seleccionat a la vista cap a una funció definida a la capa de persistència i així poder calcular el nombre d'absències existeixen la setmana passada en funció del departament mitjançant una sentència SQL.

```java
@Override
public Integer getAbsencesLastWeekDepartment(Department department) {
    Map<String, Date> lastWeekDates = calculateLastWeekDates();
    return absenceRepo.findLastWeekAbsencesByDepartment(
            lastWeekDates.get("firstDayLastWeek"),
            lastWeekDates.get("lastDayLastWeek"),
            department.getName());
}
```

Il·lustració 42: Exemple lògica negoci 2

Per concloure la capa de negoci explicaré un parell de classes necessàries per a la utilització de selectors a les vistes i que els objectes siguin enviats cap al servei de manera satisfactòria. Aquestes classes s'anomenen converters i en aquesta aplicació s'han utilitzat dos per definició dels models.



Il·lustració 43: Paquet converters

```java
@FacesConverter("departmentConverter")
@Component("departmentConverter")
public class DepartmentConverter implements Converter {

    @Autowired
    IDepartmentService departmentService;

    @Override
    public Object getAsObject(FacesContext context, UIComponent component, String value) {
        if (value == null || value.length() == 0) {
            return null;
        }
        return departmentService.findById(Integer.parseInt(value));
    }

    @Override
    public String getAsString(FacesContext context, UIComponent component, Object value) {
        if (value == null || value.equals("")) {
            return null;
        }
        return ((Department)value).getId().toString();
    }

}
```

Il·lustració 44: Converter Department

Com es pot apreciar la classe implementa la interfície Converter amb les funcions necessàries a implementar. A més per a integrar-se amb la vista de presentació es necessiten dues anotacions que són @FacesConverter i @Component:

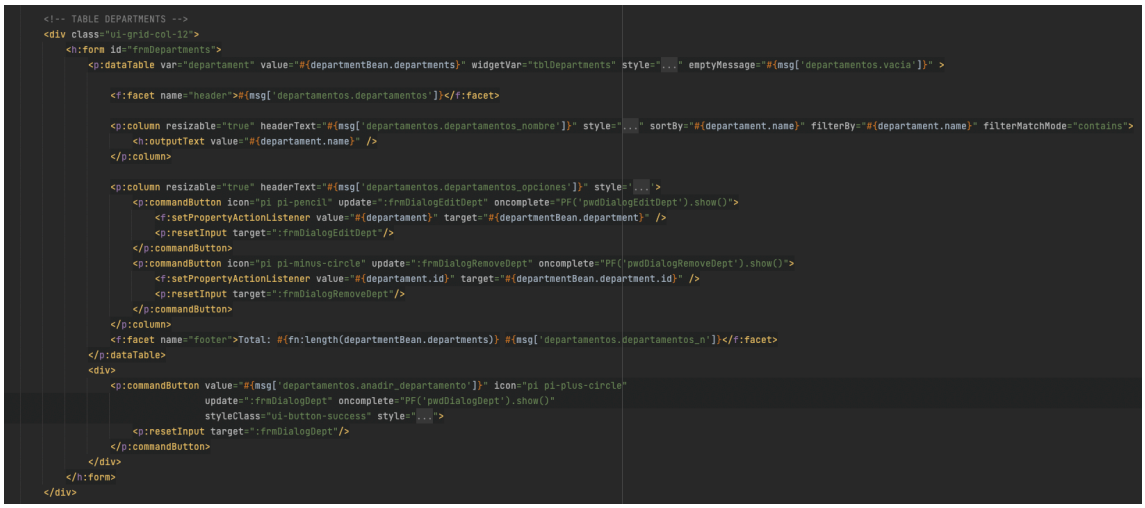
- **@FacesConverter:** Anotació per definir el nom del converter i d'aquesta manera poder utilitzar-lo a la capa de presentació al component de selecció desitjat.
- **@Component:** Anotació per definir el converter i poder carregar-lo al context de Spring.

L'ús dels converters és realment necessari per a identificar com s'envien els tipus dels selectors i d'acord amb aquests poder retornar un tipus o una altra de dades tal com es mostra a la captura anterior.

4.7. Capa de presentació

La capa de negoci és l'encarregada de presentar les vistes i plasmar la informació calculada o extreta de base de dades perquè l'usuari final ho visualitzi a la pantalla i interactuï amb la informació.

En aquesta capa primer de tot es defineix la vista la qual vol accedir l'usuari i que estarà associada a una URL dintre de l'aplicació. En aquesta explicació es mostrarà la vista de la vista de departaments tal com es mostra a la següent captura:



Il·lustració 45: Taula vista departments

A l'anterior captura trobem el codi de la taula que visualitza tots els departaments. Aquest codi conté parts que són importants i que s'expliquen en detall a continuació amb diferents captures.

Il·lustració 46: Taula departments detall 1

Aquí es carrega un atribut nomenat departaments del *bean* departmentBean i, per tant, la vista s'està comunicant amb el *backend* per a extreure la informació processada a la resta de capes. A més al final de la línia trobem un missatge de text en el cas que no existeixin resultats per al departament i que aquest serà traduït en funció de l'idioma que tingui seleccionat l'usuari.

Una altra part important de com es comunica la vista amb el *bean* és a l'hora de realitzar modificacions amb les dades per mitjà de formularis i, per tant, un exemple seria clicant en un botó d'edició del departament. A la mateixa taula trobem una columna a aquest codi:

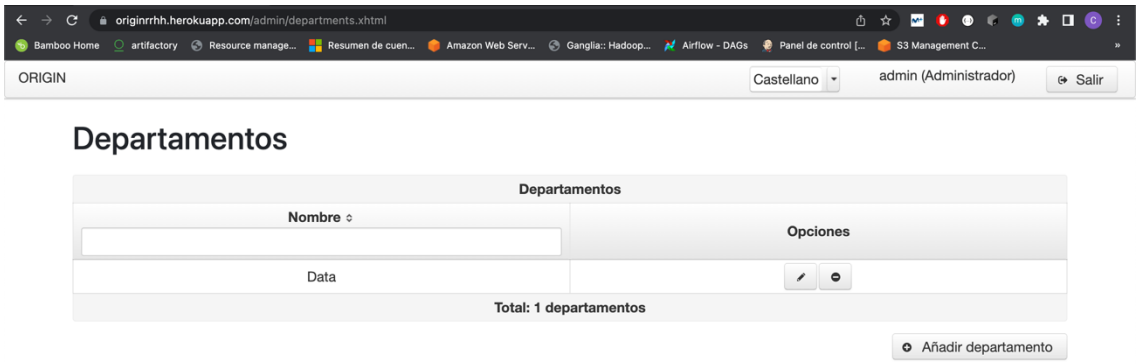
Il·lustració 47: Taula departments detall 2

Il·lustració 48: Taula departments detall 3

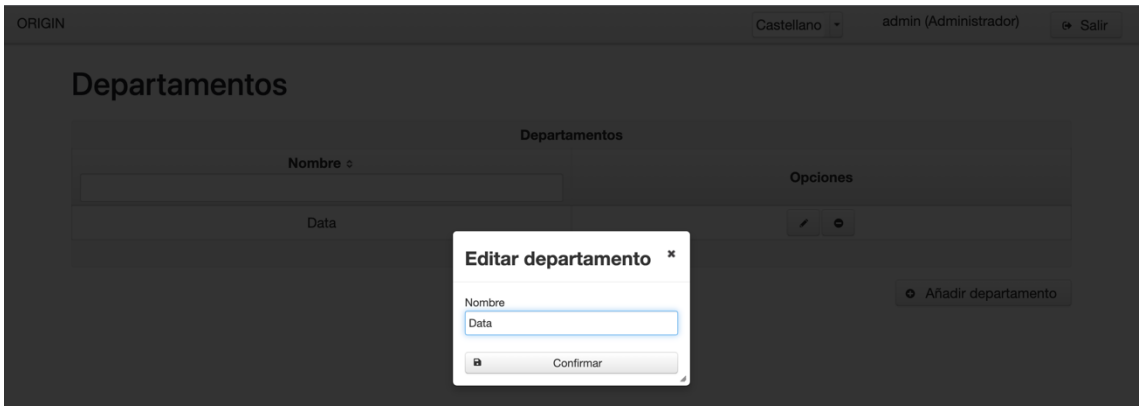
Al primer fragment s'utilitza l'etiqueta setPropertyActionListener per carregar el *bean* amb les dades del departament seleccionat i, per tant, cridar un formulari que carregui les dades del *bean* als camps corresponents.

El segon fragment de codi és el formulari d'edició del departament amb el camp d'inserció de text del nom del departament. Aquest camp tal com es pot apreciar apunta cap al departament del *bean* i en concret a l'atribut name. A més, el botó de confirmació del formulari crida a la funció editDepartment del *bean* associat i carregat i, per tant, d'aquest pot continuar el flux d'edició per les diferents capes fins a persistir l'actualització.

A continuació es mostren dues captures de la vista departaments:

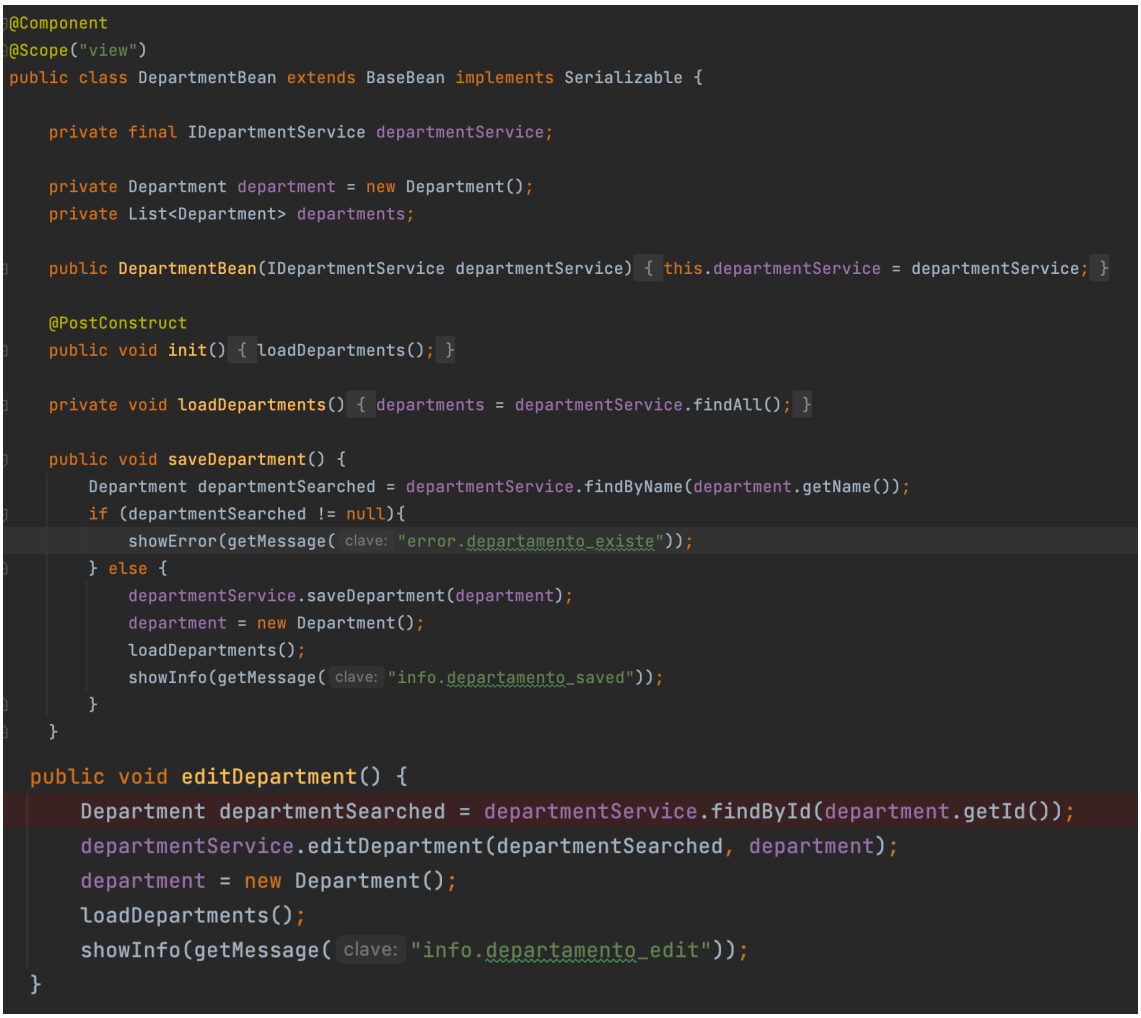


Il·lustració 49: Taula departaments UI



Il·lustració 50: Modal edició departament

Aquesta part seria la relacionada purament amb la vista que l'aplicació mostra i que l'usuari visualitza al navegador, però també falta explicar el *bean* que hem anomenat a l'explicació anterior. A continuació es mostra el *bean* associat a la vista departament:



```
@Component
@Scope("view")
public class DepartmentBean extends BaseBean implements Serializable {

    private final IDepartmentService departmentService;

    private Department department = new Department();
    private List<Department> departments;

    public DepartmentBean(IDepartmentService departmentService) { this.departmentService = departmentService; }

    @PostConstruct
    public void init() { loadDepartments(); }

    private void loadDepartments() { departments = departmentService.findAll(); }

    public void saveDepartment() {
        Department departmentSearched = departmentService.findByName(department.getName());
        if (departmentSearched != null){
            showError(getMessage( clave: "error.departamento_existe"));
        } else {
            departmentService.saveDepartment(department);
            department = new Department();
            loadDepartments();
            showInfo(getMessage( clave: "info.departamento_saved"));
        }
    }

  public void editDepartment() {
      Department departmentSearched = departmentService.findById(department.getId());
      departmentService.editDepartment(departmentSearched, department);
      department = new Department();
      loadDepartments();
      showInfo(getMessage( clave: "info.departamento_edit"));
  }
```

Il·lustració 51: *Bean* Department

En aquest cas trobem dos tipus d'anotacions, la primera seria @Component que s'ha explicat que serveix per registrar la classe al context de Spring. La segona anotació @Scope serveix per definir la durabilitat del *bean* a les vistes.

Per aquesta anotació podem trobar diferents tipus d'abasts:

- **Singleton:** És l'abast per defecte de Spring i serveix perquè totes les peticions que utilitzin aquest *bean* retornin la mateixa informació, ja que les dades estaran *cachejades*.

- **View:** L'abast de la informació es mantindrà únicament mentre que es mostri la vista associada al *bean* associat.

- **Prototype:** L'abast de la informació serà nova, ja que es crearà una nova instància del *bean* cada cop que es realitzi una petició del *bean*.

- **Request:** L'abast de la informació continguda al *bean* és reinicia, ja que es crea una nova instància del *bean* cada cop que es realitza una nova petició HTTP.

- **Session:** L'abast de la informació es mantindrà durant tota la sessió, ja que es genera una única instància per sessió.

4.8. Autenticació i gestió de rols

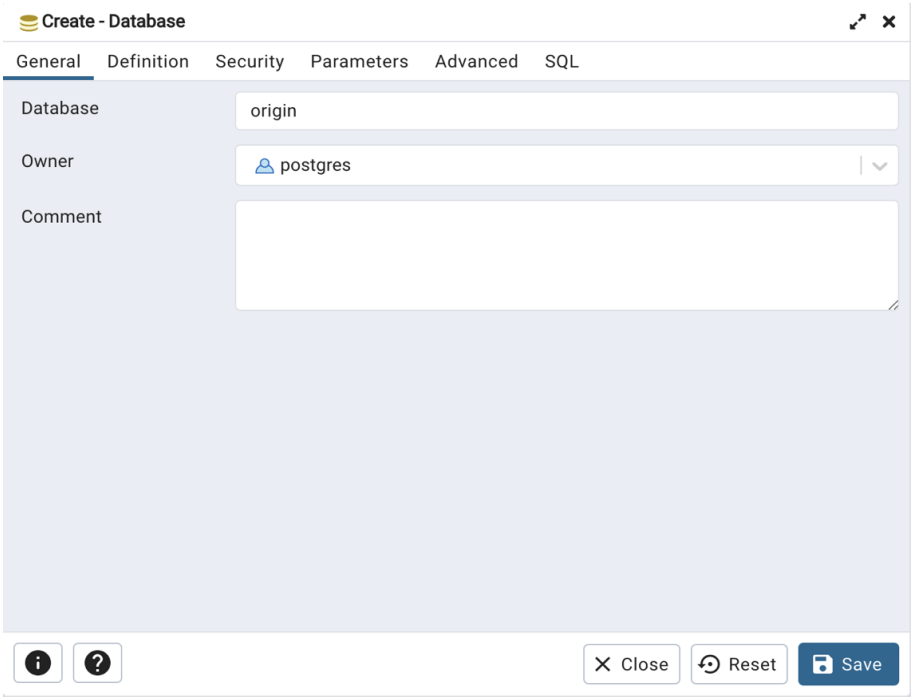
La capa de seguretat de l'aplicació està composta de l'autenticació i la gestió de rols dels usuaris per controlar els accessos a la plataforma així com els permisos a les diferents vistes.

Primer de tot, s'ha utilitzat el mòdul de seguretat spring-security per cobrir aquesta necessitat de protegir tota la plataforma. Un dels aspectes d'aquest mòdul és que solament has de configurar tota la teva seguretat mitjançant les interfícies i funcions que et proporciona el mòdul.

A continuació es detallaran cadascuna d'aquestes funcions i interfícies usades a l'aplicació. A més, es detallaran cadascuna de les estratègies per arribar a implementar la gestió de rols a la plataforma. Les classes de seguretat són les següents:

SecurityConfig:

En aquesta classe s'utilitza la funció configure(HttpSecurity http) per a definir tota mena d'autoritzacions en funció de les vistes i els rols. A continuació es mostra una captura de sobreescriptura de la funció:



Il·lustració 52: Funció configure security

La funció està composta de 4 parts conjuntes on trobem els *matchers* i els 3 *handlers*. Primer de tot els *matchers* són les regles que es defineixen per associar les vistes amb els rols per a accedir a aquestes.

A l'inici de la funció podem observar un exemple de matcher on s'indica que qualsevol usuari amb el rol ADMIN podrà accedir a qualsevol vista que contingui la ruta admin com per exemple les pantalles de departaments o usuaris. A continuació es mostra l'exemple mencionat:

```
.antMatchers("/admin/**").hasAuthority("ADMIN")
```

Il·lustració 53: Funció configure detall 1

A continuació dels *matchers* trobem una altra part important de la definició i implementació de la funció com la funcionalitat del formulari d'inici de sessió associat a la vista login.xhtml.

En aquesta part es defineix que tot usuari té permès l'accés a la vista, però s'ha utilitzat un *handler* per a definir el comportament de redireccions un cop els usuaris s'autentiquin contra l'aplicació.

A més, es defineix el cas on els usuaris no s'autentiquen correctament i, per tant, l'aplicació pot realitzar algun tipus de redirecció o emissió de missatge d'error mitjançant un altre *handler* d'inici de sessió incorrecte. A continuació es mostra la part del codi corresponent a la definició nomenada anteriorment:

```
.formLogin()
.successHandler(loginSuccessHandler())
.loginPage("/login.xhtml")
.permitAll()
.failureUrl("/login.xhtml?error=true")
.failureHandler(loginFailureHandler())
```

Il·lustració 54: Funció configure detall 2

Per finalitzar aquesta explicació, la part final de la funció tracta sobre l'accés incorrecte a vistes on els usuaris no tenen accés per les regles definides anteriorment als *matchers* i per això es crea un *handler* per generar aquesta funcionalitat i redirigir aquestes peticions indegudes a una pàgina d'error 403. A continuació es mostra el codi per definir la part mencionada anteriorment:

```
.exceptionHandling()
.accessDeniedHandler(accessDeniedHandler())
.accessDeniedPage("/accessDenied.xhtml")
```

Il·lustració 55: Funció configure detall 3

CustomAccessDeniedHandler:

Classe encarregada d'implementar la funcionalitat per aquells accessos no permesos i que implementa la interfície AccessDeniedHandler. Per tant, sobreescrivint la seva funció handle es pot obtenir la funcionalitat desitjada.

En aquest cas es busca que quan un usuari no desitjat vol accedir a una vista sense el permís corresponent llavors aquest serà redirigit a la pàgina accessDenied.xhtml tal com es mostra al següent fragment de codi:

```
public class CustomAccessDeniedHandler implements AccessDeniedHandler {
    @Override
    public void handle(HttpServletRequest request, HttpServletResponse response, AccessDeniedException exc)
            throws IOException {
        Authentication auth = SecurityContextHolder.getContext().getAuthentication();
        if (auth != null) {
            System.out.println("User: " + auth.getName()
                    + " attempted to access the protected URL: "
                    + request.getRequestURI());
        }
        response.sendRedirect( s: request.getContextPath() + "/accessDenied.xhtml");
    }
}
```

Il·lustració 56: Implementació CustomAccessDeniedHandler

LoginFailureHandler:

Classe encarregada de gestionar la funcionalitat de l'aplicació quan els usuaris s'han autenticat de manera incorrecta. Aquesta classe implementa la interfície AuthenticationFailureHandler i, per tant, si es vol desenvolupar una funcionalitat pròpia llavors, se sobreescriu la funcionalitat pròpia onAuthenticationFailure.

En aquest cas es pot observar com el codi el que realitza és una verificació pel tipus d'error que retorna l'excepció a l'hora d'iniciar sessió i si es troba el cas de credencials invàlides llavors, retorna un missatge d'error que l'usuari podrà observar a la pantalla d'inici de sessió. Al contrari, si l'error és un altre que l'anterior llavors l'aplicació mostrarà un missatge d'error d'inici de sessió incorrecte per motius inesperats.

Finalment, es redirigeix a l'usuari cap a la pàgina d'inici de sessió login.xhtml, però amb el missatge d'error que pertoqui. A continuació es mostra la classe nomenada anteriorment:

```
public class LoginFailureHandler extends BaseBean implements AuthenticationFailureHandler{

    @Autowired
    private ServletContext context;

    @Override
    public void onAuthenticationFailure(
            HttpServletRequest request,
            HttpServletResponse response,
            AuthenticationException exception)
            throws IOException, ServletException {
            if (exception.getMessage().equals("Bad credentials")){
                request.setAttribute( s: "messageError",  o: "login.malas_credenciales");
            } else {
                request.setAttribute( s: "messageError",  o: "login.inesperado");
            }
        RequestDispatcher dispatcher = context.getRequestDispatcher( s: "/login.xhtml");
        dispatcher.forward(request, response);
    }

}
```

Il·lustració 57: Implementació LoginFailureHandler

LoginSuccessHandler:

Classe encarregada de gestionar la funcionalitat de l'aplicació un cop els usuaris s'han autenticat correctament. Com es pot observar implementa la interfície SimpleUrlAuthenticationSuccessHandler i, per tant, si se sobreescriu la funció onAuthenticationSuccess llavors, es pot desenvolupar la funcionalitat de l'aplicació un cop els usuaris ingressin les credencials correctament.

En aquest cas, podem observar que qui realitza la funció és una validació dels rols que té l'usuari i si l'usuari és un candidat llavors és redirigit a la pantalla que convingui o pel contrari el podem redirigir cap a la pantalla d'index.xhtml. Exemple d'una redirecció en funció dels rols:

```java
public class LoginSuccessHandler extends SimpleUrlAuthenticationSuccessHandler {

    @Override
    public void onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse response,
                                        Authentication authentication) throws IOException {

        UserDetails userDetails = (UserDetails) authentication.getPrincipal();

        String redirectURL = request.getContextPath();

        Collection<? extends GrantedAuthority> authorities = userDetails.getAuthorities();

        for (GrantedAuthority grantedAuthority: authorities) {
            if (grantedAuthority.getAuthority().trim().equals("CANDIDATE")) {
                redirectURL = "/candidate/absences.xhtml";
            } else {
                redirectURL = "/index.xhtml";
            }
        }
        response.sendRedirect(redirectURL);

    }

}
```

Il·lustració 58: Implementació LoginSuccessHandler

A continuació es mostra una captura del codi de l'aplicació on tots els usuaris són redirigits a la pantalla d'índex i allà es controla qui té visibilitat dels enllaços cap a les diferents vistes en funció del rol:

```java
@Override
public void onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse response,
                                    Authentication authentication) throws IOException {

    String redirectURL = "/index.xhtml";
    response.sendRedirect(redirectURL);

}
```

Il·lustració 59: Funció onAuthenticationSuccess

4.9. Proves

Les proves estan definides segons els diferents casos d'ús i les respectives dependències per assolir la cobertura total de l'aplicació segons les especificacions inicials:

Cas d'ús: RF1 – Identificar-se a l'aplicació

Taula 1: Cas d'ús 1

Tester	Carlos Liarte Navarro
Propòsit	Testejar que un actor, prèviament registrat al sistema, es pot verificar al sistema.

Codi	Accions a verificar	Resultat esperat	Verificació
CU01_01	Introducció de dades d'usuari vàlides	Accés com a usuari del sistema obtingut	✓
CU01_02	Introducció de nom d'usuari incorrecte.	Missatge informant que les credencials són invàlides.	✓
CU01_03	Introducció de contrasenya d'usuari incorrecte.	Missatge informant que les credencials són invàlides.	✓
CU01_04	Introducció de nom buit.	Missatge informant que les credencials són invàlides.	✓
CU01_04	Introducció de contrasenya buida.	Missatge informant que les credencials són invàlides.	✓

Cas d'ús: RF2 – Sortir de l'aplicació

Taula 2: Cas d'ús 2

Tester	Carlos Liarte Navarro
Propòsit	Testejar que un actor pot tancar la sessió que tenia oberta al sistema.

Codi	Accions a verificar	Resultat esperat	Verificació
CU02_01	Selecció de l'opció *Logout*.	L'usuari deixa d'estar verificat al sistema.	✓

Cas d'ús: RF3 – Gestionar usuaris

Taula 3: Cas d'ús 3

Tester	Carlos Liarte Navarro
Propòsit	Testejar que un usuari administrador pugui gestionar els usuaris.

Codi	Accions a verificar	Resultat esperat	Verificació
CU03_01	Consultar la llista de tots els usuaris.	Es mostra la pantalla d'usuaris, que inclou la llista de tots els usuaris existents.	✓
CU03_02	Un cop en la pantalla de gestió d'usuaris, crear un nou usuari, introduint els camps requerits.	Es crea l'usuari i es mostra a la llista d'usuaris de la pantalla de gestió d'usuaris.	✓
CU03_03	Editar un usuari.	Es modifica l'usuari i es mostra a la llista d'usuaris de la pantalla de gestió d'usuaris amb les noves dades.	✓
CU03_04	Eliminar un usuari.	S'elimina l'usuari i es deixa de mostrar a la llista d'usuaris de la pantalla de gestió d'usuaris.	✓
CU03_05	Desactivar un usuari.	Es mostra un missatge informant que l'usuari està desactivat i es mostra l'usuari a la llista disponible per habilitar.	✓
CU03_05	Crear un usuari amb el mateix DNI d'un usuari ja existent.	Missatge informant que ja existeix un usuari amb aquest DNI.	✓

Cas d'ús: RF4 – Gestionar candidats

Taula 4: Cas d'ús 4

Tester	Carlos Liarte Navarro
Propòsit	Testejar que un usuari de recursos humans pugui gestionar els candidats.

Codi	Accions a verificar	Resultat esperat	Verificació
CU04_01	Consultar la llista de tots els candidats.	Es mostra la pantalla de recursos humans, que inclou la llista de tots els candidats existents.	✓
CU04_02	Crear un nou candidat, introduint els camps requerits.	Es crea el candidat i es mostra a la llista de candidats.	✓

CU04_03	Editar un candidat.	Es modifica el candidat i es mostra a la llista de candidats amb les noves dades.	✓
CU04_04	Eliminar un candidat.	S'elimina el candidat i es deixa de mostrar a la llista de candidats.	✓
CU04_05	Desactivar un candidat.	El candidat està desactivat i es mostra l'usuari a la llista disponible per habilitar.	✓
CU04_06	Consultar la informació del candidat.	Es mostra la pantalla d'informació del candidat, que inclou tota la informació del candidat.	✓
CU04_07	Contactar amb el candidat.	Missatge informant que el missatge s'ha enviat correctament al candidat.	✓

Cas d'ús: RF5 – Visualitzar posicions obertes per departament

Taula 5: Cas d'ús 5

Tester	Carlos Liarte Navarro
Propòsit	Testejar que un actor de recursos humans pot filtrar per departament a la llista de posicions obertes.

Codi	Accions a verificar	Resultat esperat	Verificació
CU05_01	Consultar la llista de posicions obertes.	S'accedeix a la pantalla de recursos humans i es mostra la llista de posicions obertes.	✓
CU05_02	Introduir el nom del departament a cercar.	Es filtra la llista de posicions obertes i apareixen aquelles associades al departament cercat.	✓

Cas d'ús: RF6 – Visualitzar absències

Taula 6: Cas d'ús 6

Tester	Carlos Liarte Navarro
Propòsit	Testejar que un actor de recursos humans pot visualitzar les absències.

Codi	Accions a verificar	Resultat esperat	Verificació
CU06_01	Consultar la llista d'absències.	S'accedeix a la pantalla de visualització d'absències.	✓
CU06_02	Introduir el departament a cercar les absències.	Es mostra la llista d'absències actuals i futures per aquell department.	✓

Cas d'ús: RF7 – Contactar candidats

Taula 7: Cas d'ús 7

Tester	Carlos Liarte Navarro
Propòsit	Testejar que un usuari de recursos humans pugui contactar amb un candidat.

Codi	Accions a verificar	Resultat esperat	Verificació
CU07_01	Consultar la llista de tots els candidats.	Es mostra la pantalla de recursos humans, que inclou la llista de tots els candidats existents.	✓
CU07_02	Introduir el missatge buit.	Missatge informant que hi ha un error de validació.	✓
CU07_02	Enviar el missatge omplert.	Missatge informant s'ha enviat el missatge correctament.	✓

Cas d'ús: RF8 – Gestionar absències

Taula 8: Cas d'ús 8

Tester	Carlos Liarte Navarro
Propòsit	Testejar que un manager pot gestionar les absències.

Codi	Accions a verificar	Resultat esperat	Verificació
CU08_01	Consultar la llista de totes les absències dels treballadors pendents d'aprovació del manager.	Es mostra la pantalla d'absències per aprovar ,que inclou les absències que demana un treballador al seu mànager.	✓
CU08_02	S'accepta una absència.	S'actualitzen la llista d'absències del treballador i el manager amb l'estat modificat a acceptat.	✓
CU08_03	Es declina una absència.	S'actualitzen la llista d'absències del treballador i el manager amb l'estat modificat a declinat.	✓

Cas d'ús: RF9 – Sol·licitar absència

Taula 9: Cas d'ús 9

Tester	Carlos Liarte Navarro
Propòsit	Testejar que un treballador pot sol·licitar una absència.

Codi	Accions a verificar	Resultat esperat	Verificació
CU09_01	Consultar la llista de totes les absències d'un treballador.	Es mostra la pantalla d'absències ,que inclou les absències que ha demanat un treballador.	✓
CU09_02	Crear una nova absència.	Es mostra un missatge informant que s'ha creat correctament i s'actualitza la taula amb la nova absència en estat pendent.	✓

Cas d'ús: RF10 – Gestionar dades personals

Taula 10: Cas d'ús 10

Tester	Carlos Liarte Navarro
Propòsit	Testejar que un usuari que no sigui administrador pot gestionar les dades del perfil.

Codi	Accions a verificar	Resultat esperat	Verificació
CU10_01	Consultar el perfil de l'usuari.	Es mostra la pantalla de perfil de l'usuari verificat pel sistema.	✓
CU10_02	Editar el perfil de l'usuari.	Es mostra un missatge informant que l'usuari s'ha editat correctament i s'actualitzen les dades de la taula.	✓

Cas d'ús: RF11 – Gestionar candidatures

Taula 11: Cas d'ús 11

Tester	Carlos Liarte Navarro
Propòsit	Testejar que un usuari de recursos humans pot gestionar les candidatures.

Codi	Accions a verificar	Resultat esperat	Verificació
CU11_01	Consultar la llista de candidatures.	Es mostra la pantalla de candidatures amb la llista de candidatures.	✓
CU11_02	Accedir a una posició oberta.	Es mostra la pantalla d'informació d'una posició oberta amb la llista de candidats, que seran les candidatures.	✓
CU11_03	Crear una candidatura.	Es mostra el candidat a la llista de candidats de la posició oberta que a la qual es presenta.	✓

Cas d'ús: RF12 – Visualitzar estat candidatura

Taula 12: Cas d'ús 12

Tester	Carlos Liarte Navarro
Propòsit	Testejar que un usuari de recursos humans pot visualitzar l'estat les candidatures.

Codi	Accions a verificar	Resultat esperat	Verificació
CU12_01	Consultar la llista de candidatures.	Es mostra la pantalla de candidatures amb la llista de candidatures.	✓
CU12_02	Consultar la candidatura.	Es mostra la pantalla d'informació de la candidatura amb el respectiu estat i comentaris pertinents.	✓

Cas d'ús: RF13 – Canviar estat candidatura

Taula 13: Cas d'ús 13

Tester	Carlos Liarte Navarro
Propòsit	Testejar que un usuari de recursos humans pot modificar l'estat les candidatures.

Codi	Accions a verificar	Resultat esperat	Verificació
CU13_01	Consultar llista de posicions obertes.	Es mostra la pantalla de posicions obertes amb la llista de posicions obertes.	✓
CU13_02	Accedir informació posició oberta.	Es mostra la pantalla de detall de la posició oberta.	✓
CU13_03	Modificar estat candidatura.	Es mostra un missatge informant que l'estat de la candidatura s'ha canviat exitosament i s'envia correu electrònic al candidat amb el canvi d'estat.	✓

Cas d'ús: RF14 – Assignar posició oberta a departament

Taula 14: Cas d'ús 14

Tester	Carlos Liarte Navarro
Propòsit	Testejar que un usuari de recursos humans pot assignar la posició oberta a un departament.

Codi	Accions a verificar	Resultat esperat	Verificació
CU14_01	Consultar llista de posicions obertes.	Es mostra la pantalla de posicions obertes amb la llista de posicions obertes.	✓
CU14_02	Llistar els departaments disponibles.	Es mostra un desplegable al formulari d'edició amb la llista de departaments creats per assignar.	✓
CU14_03	Introduir el departament nou en la edició de la posició oberta.	Es mostra un missatge informant que la posició oberta s'ha modificat correctament i s'actualitza la llista de posicions obertes.	✓

Cas d'ús: RF15 – Gestionar departaments

Taula 15: Cas d'ús 15

Tester	Carlos Liarte Navarro
Propòsit	Testejar que un usuari administrador pot gestionar els departaments.

Codi	Accions a verificar	Resultat esperat	Verificació
CU15_01	Consultar llista de departaments.	Es mostra la pantalla de departaments amb la llista de tots els departaments.	✓
CU15_02	Introduir el nom del departament buit.	Es mostra un missatge d'error de validació.	✓
CU15_03	Introduir el nom d'un departament	Es mostra un missatge d'error informant	

	ja existent.	que ja existeix el departament.	✓
CU15_04	Introduir un nom de departament nou.	Es mostra un missatge informant que s'ha creat el departament i s'actualitza la llista de departaments.	✓
CU15_05	Editar departament.	Es mostra un missatge informant que el departament s'ha editat correctament i s'actualitza la llista amb les noves dades.	✓
CU15_06	Esborrar departament.	Es mostra un missatge informant que s'ha esborrat correctament el departament i s'actualitza la llista eliminant el departament seleccionat.	✓

Cas d'ús: RF16 – Gestionar posicions obertes

Taula 16: Cas d'ús 16

Tester	Carlos Liarte Navarro
Propòsit	Testejar que un usuari de recursos humans pot gestionar les posicions obertes.

Codi	Accions a verificar	Resultat esperat	Verificació
CU16_01	Consultar la llista de posicions obertes.	Es mostra la pantalla de posicions obertes, que inclou la llista de totes les posicions obertes existents.	✓
CU16_02	Crear una nova posició oberta, introduint els camps requerits.	Es mostra un missatge informant que la posició oberta s'ha creat correctament i s'actualitza la llista de posicions obertes.	✓
CU16_03	Editar una posició oberta.	Es mostra un missatge informant que s'ha editat correctament i s'actualitza la llista de posicions obertes amb les noves dades.	✓
CU16_04	Eliminar una posició oberta.	Es mostra un missatge informant que s'ha esborrat correctament i s'actualitza la llista de posicions obertes sense afegir la esborrada.	✓

CU16_05	Consultar el detall de la posició oberta.	Es mostra la pantalla de detall de la posició oberta amb tota la informació i la llista de candidats associats.	✓
CU16_06	Consultar la informació del candidat.	Es mostra la pantalla d'informació del candidat, que inclou tota la informació del candidat.	✓

Cas d'ús: RF17 – Assignar candidat a posició oberta

Taula 17: Cas d'ús 17

Tester	Carlos Liarte Navarro
Propòsit	Testejar que un usuari de recursos humans assignar un candidat a una posició oberta.

Codi	Accions a verificar	Resultat esperat	Verificació
CU17_01	Consultar llista de posicions obertes.	Es mostra la pantalla de posicions obertes amb la llista de posicions obertes.	✓
CU17_02	Accedir informació posició oberta.	Es mostra la pantalla de detall de la posició oberta.	✓
CU17_03	Introduir el DNI del candidat a afegir a la candidatura.	Es mostra un missatge informant que el candidat s'ha afegit correctament i s'actualitza la llista de candidats. A més, s'envia un correu electrònic al candidat.	✓

5. Conclusions

En aquest treball de fi de grau s'han après algunes lliçons molt importants a l'hora de dur a terme un projecte des de zero i que en un principi no li donava molt èmfasi.

La primera de les lliçons és el valor que porta una bona planificació a l'hora de realitzar una bona planificació on comptabilitzes les tasques en funció de les fites d'entrega per a treballar d'una manera òptima i, d'aquesta manera tenir un control exhaustiu de l'estat de les tasques en tot moment.

Una segona lliçó apresa durant el projecte és la importància de la definició dels diagrames i com afecten aquests a un desenvolupament més efectiu del codi, ja que d'aquesta manera evites portar a cap canvis que modifiquen l'estructura del codi per complet. A més, la definició dels requisits implica conèixer en tot moment l'abast del projecte i, per tant, poder separar en mòduls l'aplicació per iterar de manera constant la plataforma.

L'última lliçó apresa representa el fet d'elaborar el treball sobre la metodologia àgil ,ja que comporta que cada requisit funcional de l'aplicació correspon a un cicle de desenvolupament i, en conseqüència, el codi estarà alineat amb la definició dels requeriments inicials passant cadascun d'ells per les pertinents proves.

D'aquesta manera s'arriba a una aplicació amb una cobertura òptima per arribar a un entorn productiu de manera eficient cada cop que es requereixi desenvolupar una nova funcionalitat sobre la plataforma.

Els objectius plantejats a l'inici del projecte s'han assolit correctament ja que es va realitzar un estudi previ de l'abast i, per tant, es van retallar alguns objectius que es volien realitzar durant el desenvolupament però sent realistes es van ometre. En conseqüència, els objectius que no que van quedar fora entrarien a una segona fase d'iteració de desenvolupament de la plataforma per ajudar gestionar d'una manera més eficient les posicions d'una empresa.

La planificació s'ha seguit d'una manera satisfactòria respectant les fites d'entrega per tal de no sobreposar tasques i d'aquesta manera poder desenvolupar el codi d'una manera eficient i, per tant, entregar valor amb cada pujada a l'entorn productiu.

Respecte a la metodologia emprada es pot dir que ha estat l'adequada, ja que utilitzant desenvolupament àgil es detecten problemes més ràpids, a més que el producte s'apropa als requisits inicials, perquè es troba en una validació constant amb cada nova funcionalitat que s'implementa.

Com bé he explicat, una metodologia àgil implica que existeixin canvis constants al projecte i, per exemple, la versió inicial del JDK era la 8, però per motius de bugs descoberts sobre aquesta versió referents a la conversió de caràcters especials en UTF-8 llavors es va haver de canviar a la versió 11 on s'havien corregit.

A més al diagrama UML es van definir algunes classes que amb el desenvolupament del codi van partir-se generant d'aquesta manera noves entitats no redactades inicialment, però que aportaven valor a la plataforma. Tots aquests canvis entre altres van realitzar-se per garantir l'èxit del projecte.

Finalment, aquest projecte necessita evolucionar, però per motius de temps i, per tant, aquestes han quedat en un segon pla, però no significa que tinguin menys importància que la resta desenvolupada durant el transcurs del semestre.

A continuació es mostraran les funcionalitats desitjades per a futurs desenvolupaments:

- Generar una base de dades pròpia amb informació actualitzada sobre els candidats per a gestionar futures posicions obertes dintre de l'empresa.

- Crear un sistema de publicacions públiques perquè els candidats puguin veure i inscriure's de forma automàtica a les posicions obertes oferires per l'empresa.

- Publicar les posicions obertes de manera senzilla i automàtica a les diferents plataformes com per exemple Linkedin o Infojobs entre altres i redirigir el tràfic de candidats cap a la plataforma.

- Integrar les notificacions de la plataforma a altres tecnologies com podria ser SMS o Slack entre d'altres.

- Afegir la necessitat de visualitzar els OKR de cada equip així com l'organigrama de l'empresa en diferents nivells.

6. Glossari

Posició oberta: Oferta de treball publicada internament a l'aplicació

Candidat: Persona contactada i inserida a la plataforma per poder aplicar a les diferents posicions de obertes.

Candidatura: Acció que es dona quan s'afegeix un candidat a una posició oberta. Aquesta candidatura es la que farà referencia al procés de selecció.

OKR: Acrònim de *Objective Key Results*. Significa els objectius definits a complir per a un equip o companyia en un determinat rang de temps.

7. Bibliografia

- [1] Pàgina gestió de projectes Monday [consultada: 01 de Març de 2022] https://monday.com/lang/es/
- [2] Pàgina creació de diagrames UML [consultada: 04 de Març de 2022] https://staruml.io/
- [3] Pàgina de creació de diagrames DrawIO [consultada 11 de Març de 2022] https://app.diagrams.net/
- [4] Pàgina de documentació de Spring [consultada 02 d'Abril de 2022] https://spring.io/projects/spring-boot
- [5] Pàgina de documentació de Maven [consultada 02 d'Abril de 2022] https://maven.apache.org/
- [6] Pàgina de documentació de Primefaces [consultada 10 d'Abril de 2022] https://www.primefaces.org/showcase-v8/
- [7] Pàgina descàrrega JDK 11 [consultada 28 de Març de 2022] https://www.oracle.com/es/java/technologies/javase/jdk11-archive-downloads.html
- [8] Pàgina descàrrega IntelliJ [consultada 28 de Març de 2022] https://www.primefaces.org/showcase-v8/
- [9] Pàgina descàrrega PostgreSQL [consultada 28 de Març de 2022] https://www.primefaces.org/showcase-v8/
- [10] Pàgina descàrrega Git [consultada 28 de Març de 2022] https://git-scm.com/download/mac
- [10] Pàgina repositori GitHub [consultada 28 de Març de 2022] https://github.com/cliartenuoc/origin
- [10] Pàgina documentació Heroku [consultada 15 de Maig de 2022] https://devcenter.heroku.com/categories/reference

8. Annex 1: Guia d'instal·lació d'entorn

8.1. Instal·lació JDK:

1. Descarregar la versió 11 del JDK de Java que la trobarem al següent enllaç (Es requereix compte gratuït d'Oracle per descarregar):

   https://www.oracle.com/es/java/technologies/javase/jdk11-archive-downloads.html

2. Executar el fitxer descarregat per iniciar la instal·lació:

Il·lustració 60: Instal·lació Java macOS

3. Cliquem "Continuar" i deixem totes les opcions per defecte.

4. Un cop finalitzada la instal·lació millor reiniciar la màquina per evitar problemes d'actualitzacions de directoris.

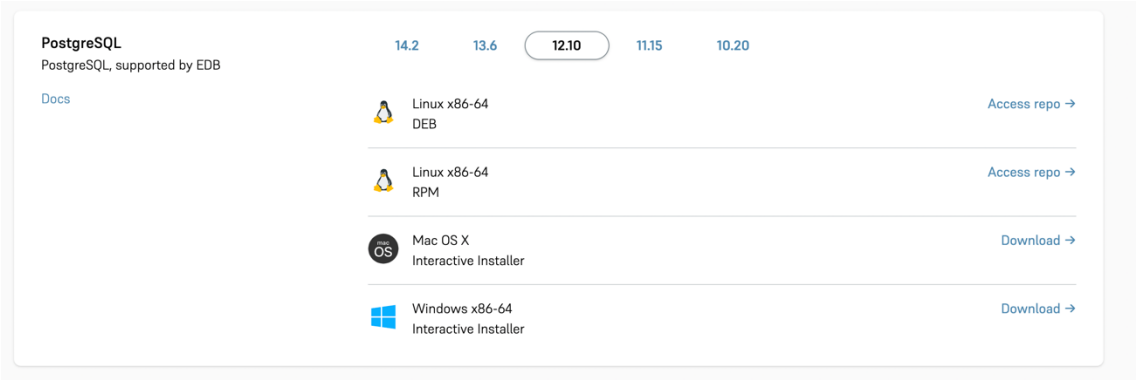
5. Validem que tenim la versió correcta instal·lada utilitzant la següent comanda al terminal:

   ```
   $ java -version
   ```

8.2. Instal·lació PostgreSQL:

1. Descarreguem PostgreSQL 12.10 del següent enllaç:

   https://www.enterprisedb.com/software-downloads-postgres

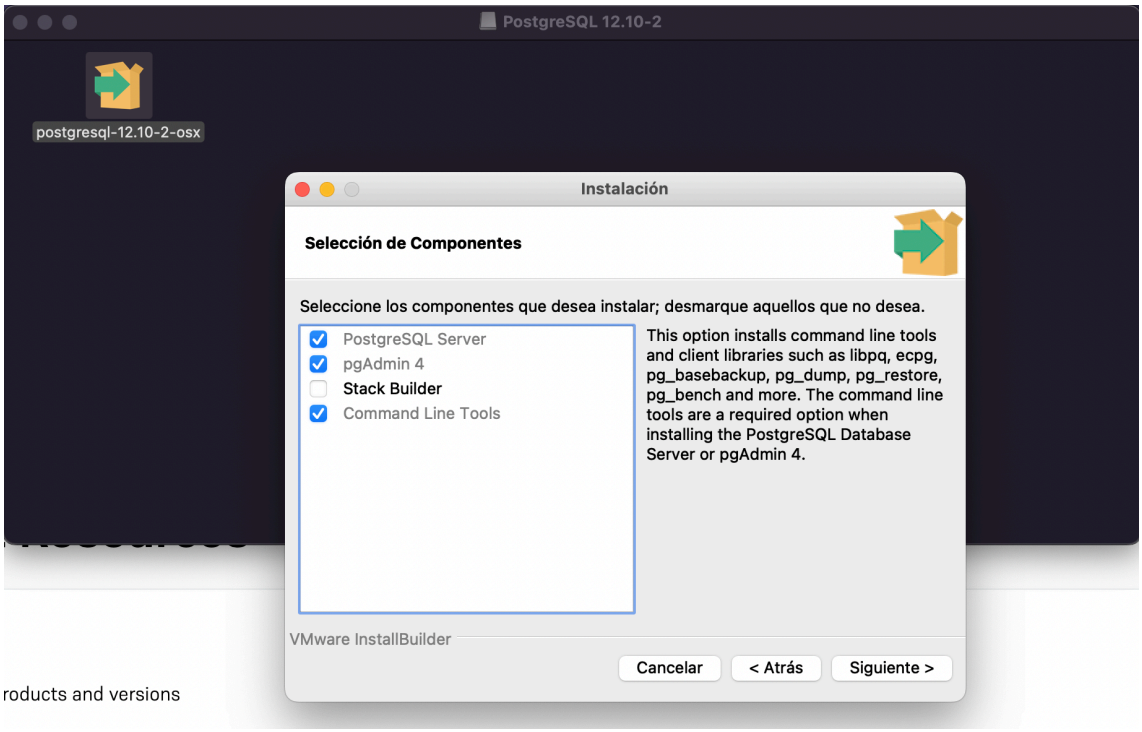


Il·lustració 61: Descàrrega PostgreSQL

2. Cliquem a l'opció adient per al nostre sistema operatiu i la descarreguem (Es necessita crear un compte gratuït):

3. Executem el fitxer descarregat per iniciar la instal·lació.

4. Seleccionem els valors indicats a la següent imatge:



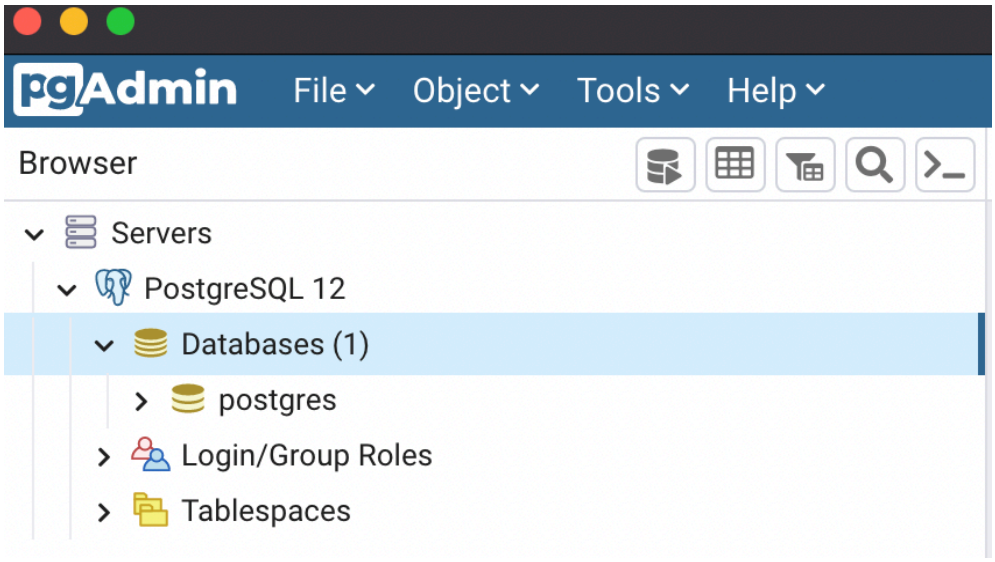
Il·lustració 62: Instal·lació PostgreSQL

5. Cliquem a següent amb els valors per defecte i reiniciem l'ordinador.

8.3. Configuració PostgreSQL:

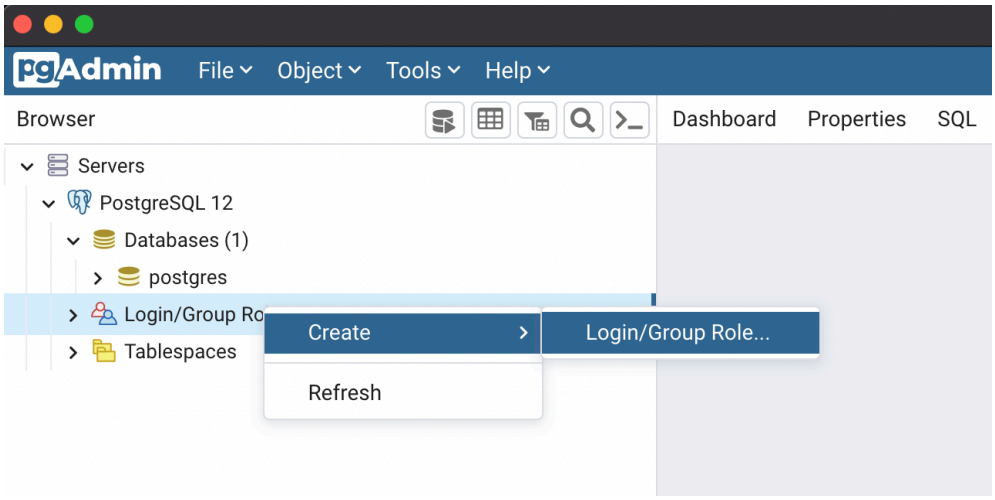
1. Obrim PgAdmin4.

2. Accedim al servidor PostgreSQL 12 fent doble clic i inserim la contrasenya de superusuari definida durant la instal·lació:



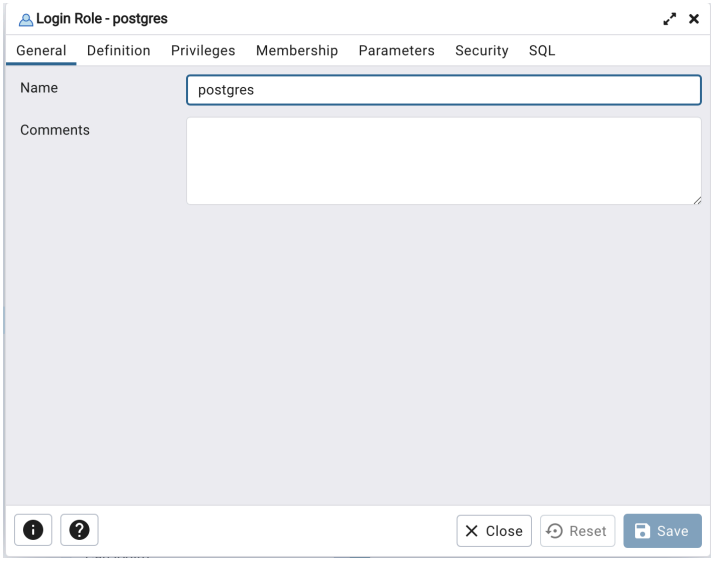
Il·lustració 63: Accés server PostgreSQL

3. Un cop connectat al servidor llavors creem un usuari per tal de no accedir com a superusuari a l'hora de treballar amb la base de dades:

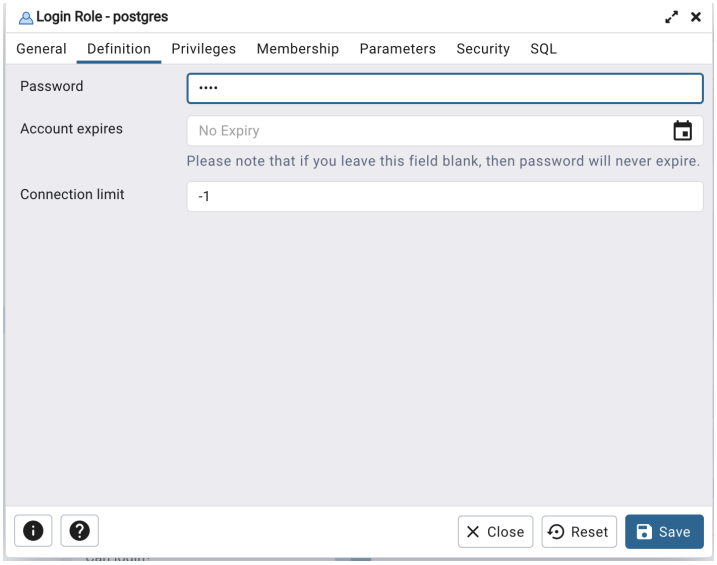


Il·lustració 64: Creació usuari Postgres 1

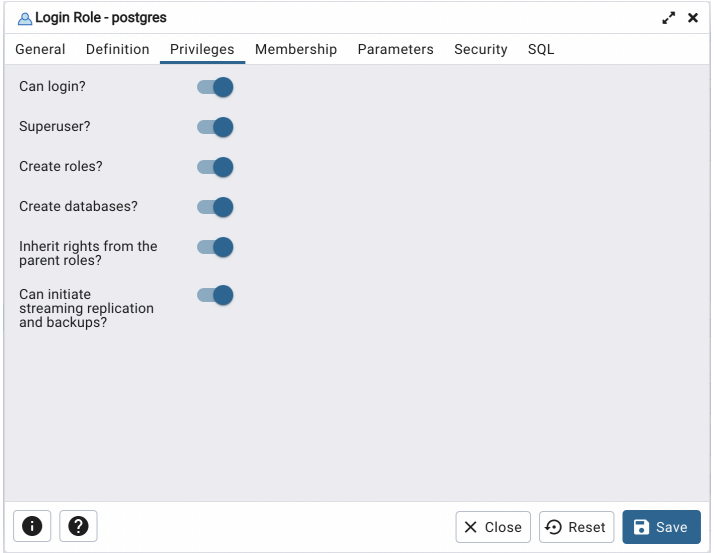
4. Creem l'usuari amb el nom "postgres" i la contrasenya "1234" amb tota mena de privilegis. Aquest usuari serà el que tindrà l'aplicació per a realitzar les consultes contra la base de dades.



Il·lustració 65: Creació usuari Postgres 2

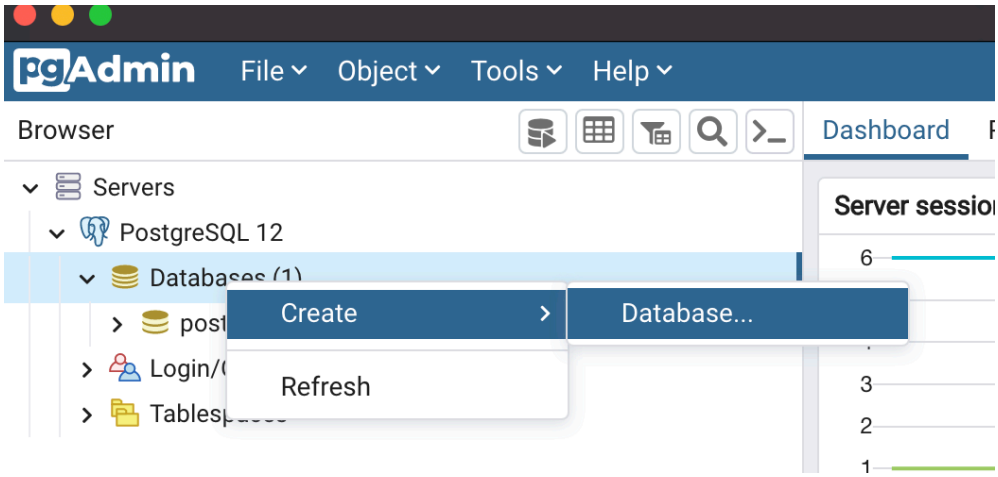


Il·lustració 66: Creació usuari Postgres 3



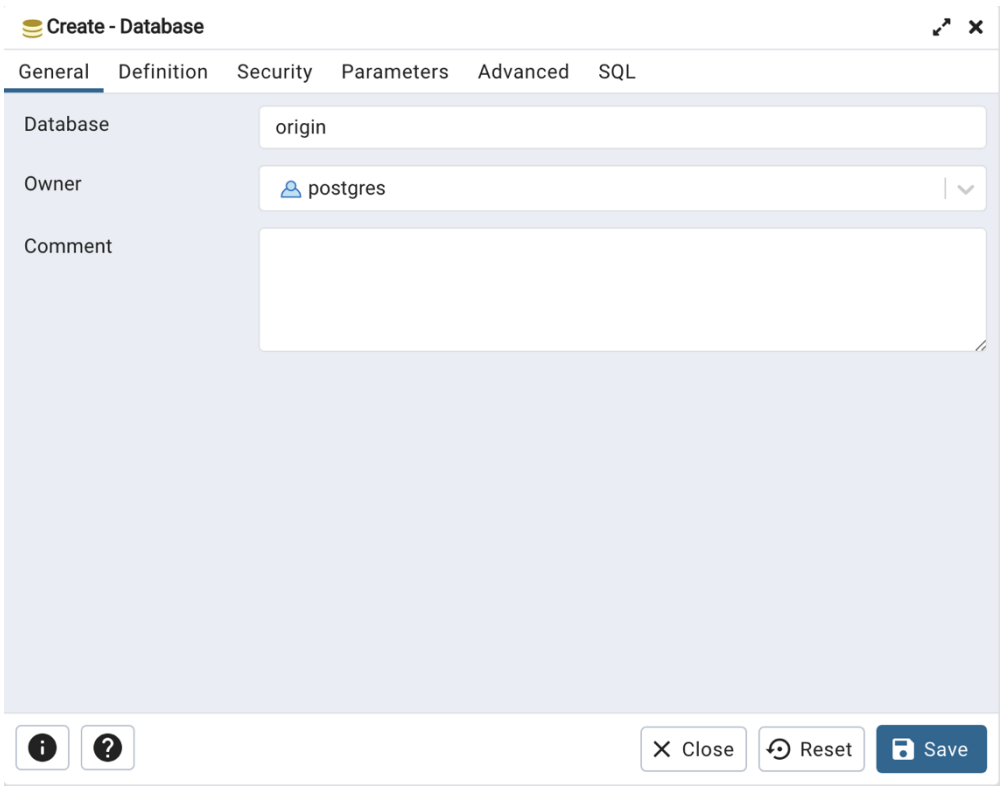
Il·lustració 67: Creació usuari Postgres 4

5. Per finalitzar la configuració del PostgreSQL creem la base de dades de l'aplicació:



Il·lustració 68: Creació base de dades Origin 1

6. Com a nom de la base de dades fiquem "origin" i assignem el propietari a l'usuari creat anteriorment "postgres":



Il·lustració 69: Creació base de dades Origin 2

8.4. Instal·lació Maven:

1. Executar la comanda següent per instal·lar Maven (Opció per mac):

   $ brew install maven

2. Un cop instal·lat Maven a l'ordinador es pot comprovar amb la següent comanda:

```
(* |mediabuying-sbx:fargate) cliarte@LFVFF72PNQ05N ~ mvn -version
Apache Maven 3.6.3 (cecedd343002696d0abb50b32b541b8a6ba2883f)
Maven home: /usr/local/Cellar/maven/3.6.3_1/libexec
```

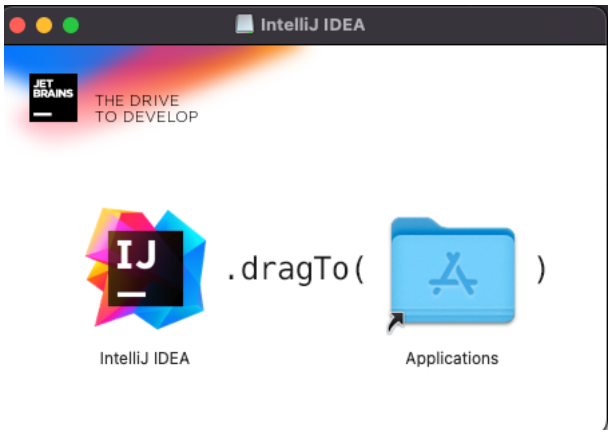
Il·lustració 70: Comprovació Maven

8.5. Descàrrega IntelliJ IDEA:

1. Descarreguem l'ide IntelliJ del següent enllaç:

   https://www.jetbrains.com/idea/download/#section=mac

2. Executem el fitxer descarregat per iniciar la instal·lació (Opció per mac).

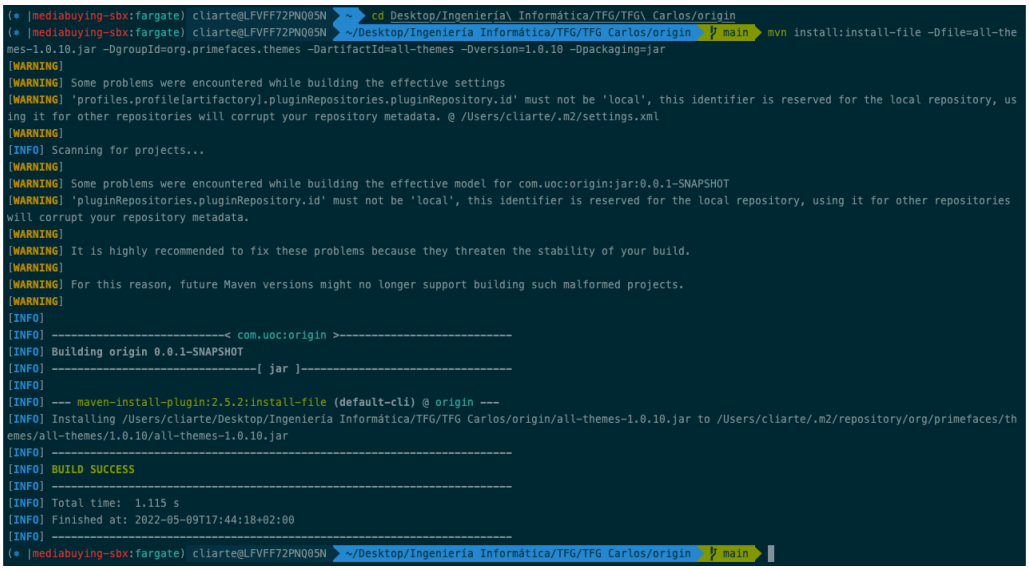


Il·lustració 71: Instal·lació IntelliJ

3. Seleccionem la ruta d'espai de treball per defecte i ja podrem iniciar-lo.

8.6. Instal·lació llibreria temes Primefaces.

1. Accedir a la carpeta del projecte un cop el tinguem descarregat via terminal i executar la següent comanda per instal·lar-la:



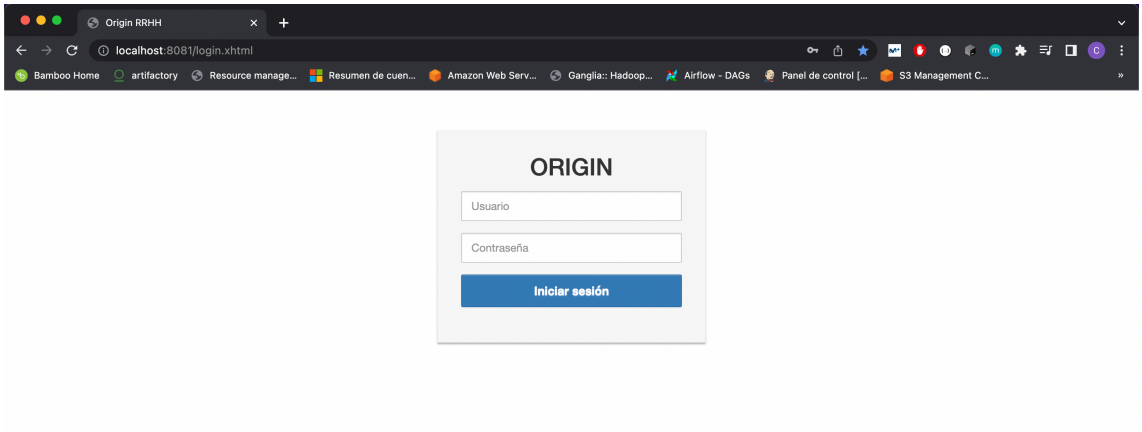
Il·lustració 72: Instal·lació Primefaces

9. Annex 2: Manual d'usuari

Aquest manual d'usuari està redactat per facilitar la utilització de la plataforma en funció dels rols d'usuari.

9.1. Inici de sessió

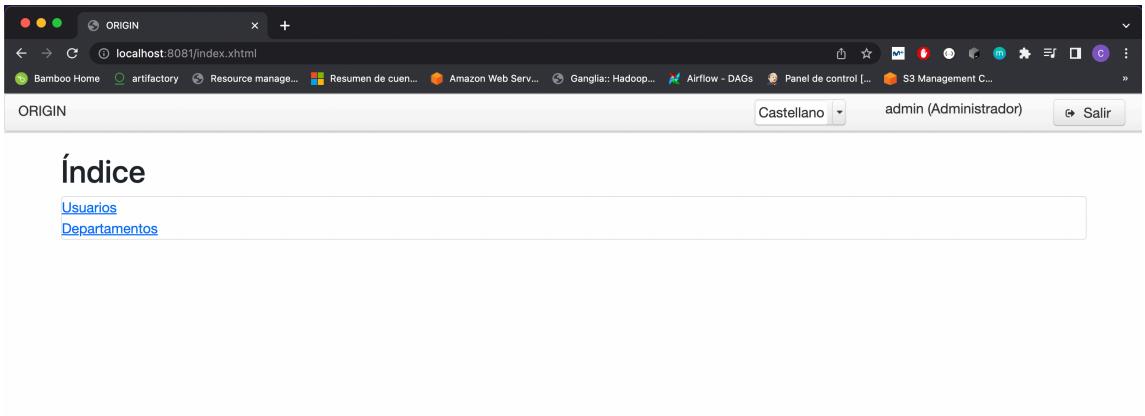
La pantalla d'inici de sessió tindrà dos camps a omplir que seran les mateixes credencials de l'usuari. Tots els usuaris accediran a la plataforma a través del seu DNI i la seva contrasenya menys l'administrador que utilitzarà les credencials "admin" com a usuari i "1234" com a contrasenya.



Il·lustració 73: Pantalla Inici sessió

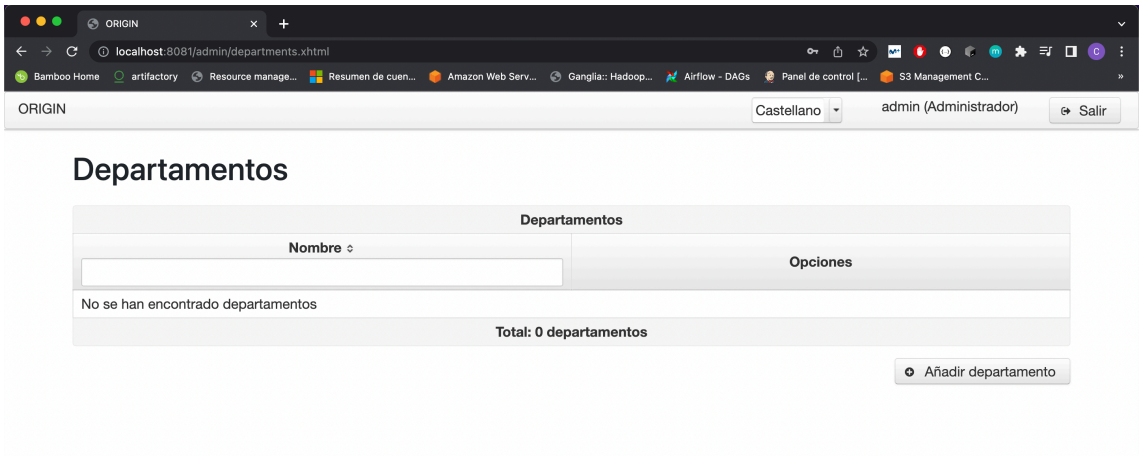
9.2. Administrador

L'administrador accedirà a la pantalla índex on trobarà els respectius enllaços a la gestió d'usuaris i departaments tal com es mostra a la següent captura:

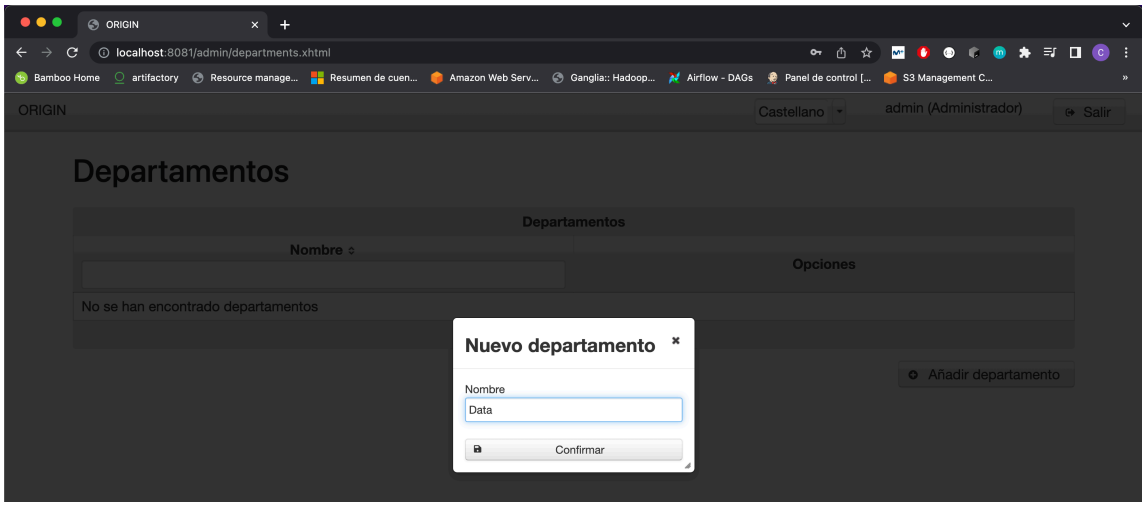


Il·lustració 74: Pantalla Índex administrador

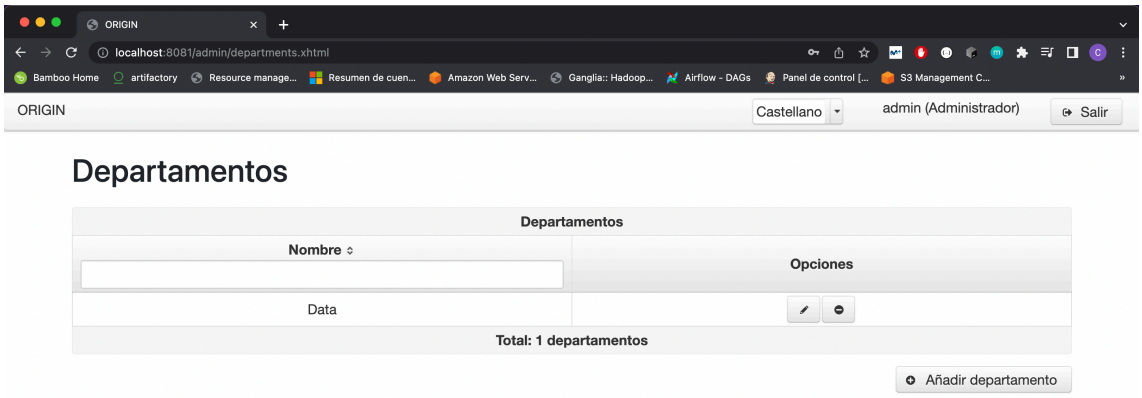
A la pantalla de gestió de departaments l'usuari administrador pot crear departaments utilitzant el respectiu formulari:



Il·lustració 75: Pantalla gestió departaments 1

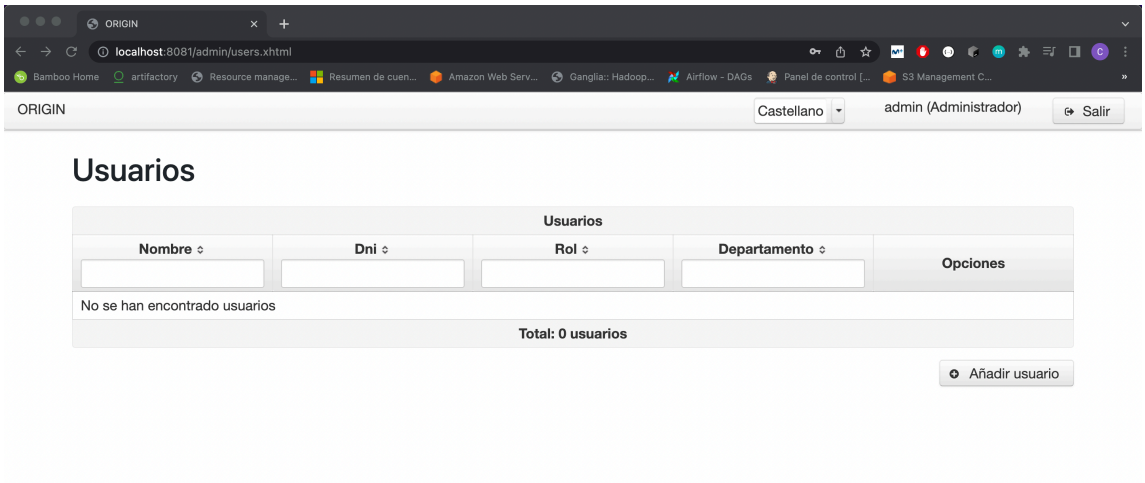


Il·lustració 76: Formulari departaments



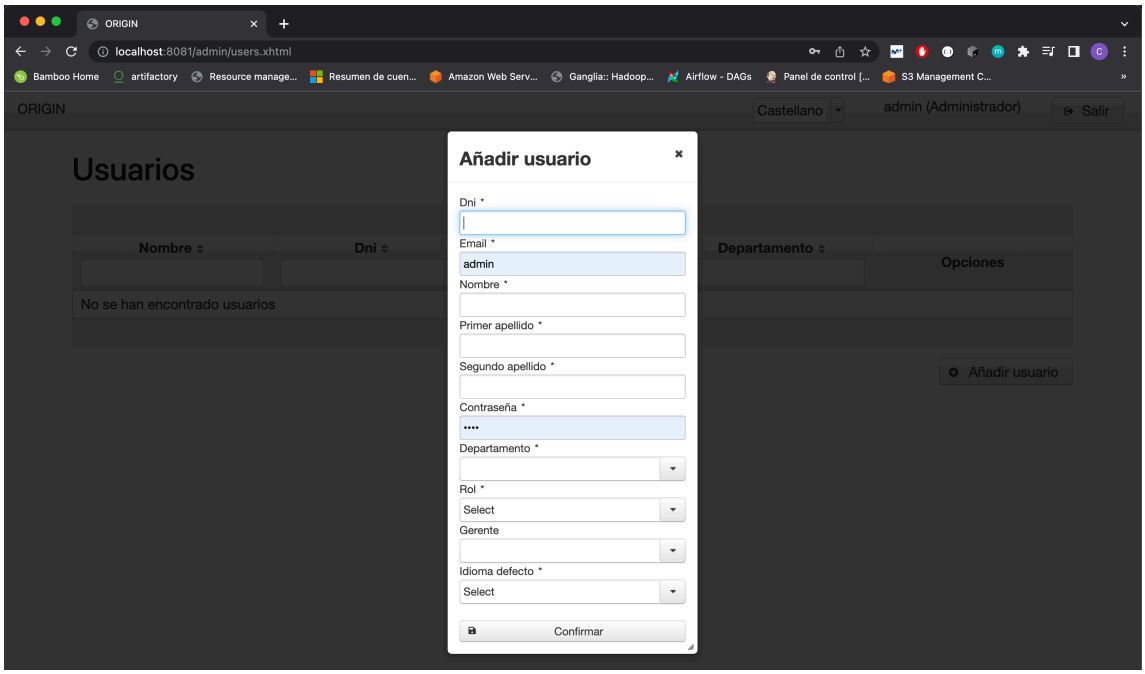
Il·lustració 77: Pantalla gestió departaments 2

A la pantalla de gestió d'usuaris l'administrador podrà veure una taula amb la llista de tots els usuaris de la plataforma on podrà filtrar per qualsevol dels camps de la taula per cercar els usuaris d'una manera més eficient.



Il·lustració 78: Pantalla gestió usuaris 1

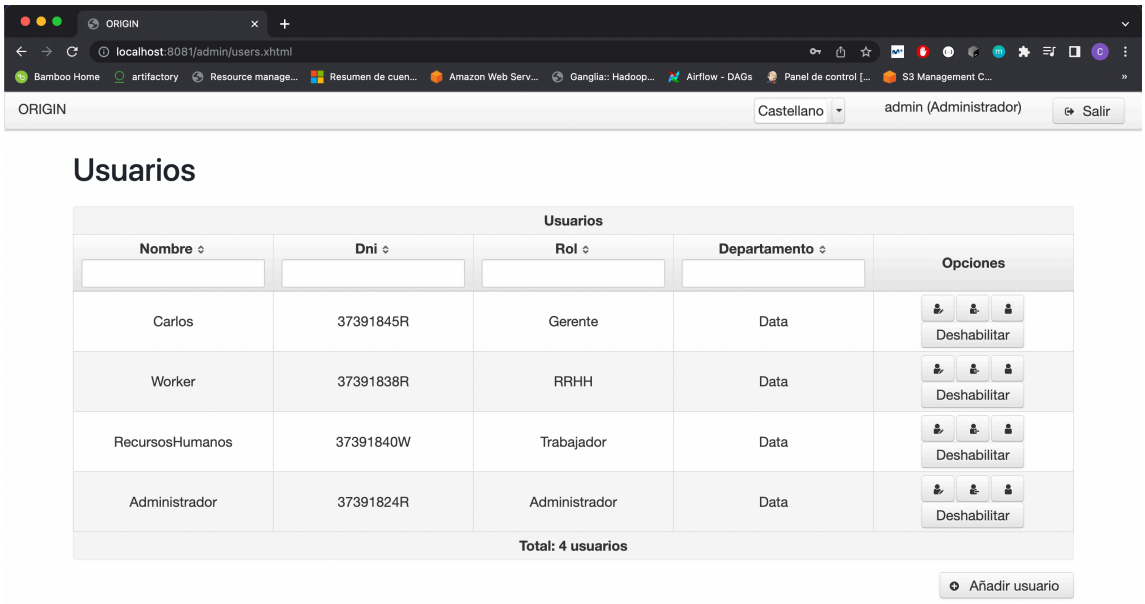
En aquesta pantalla es poden crear tants usuaris com vulgui l'administrador sempre que no existeixi un usuari amb el mateix DNI mitjançant el respectiu formulari que apareix si cliquem "Añadir usuario":



Il·lustració 79: Pantalla gestió usuaris formulari

Al formulari anterior s'afegeix la informació de l'usuari així com el departament, gerent si no es vol crear un gerent i el llenguatge predefinit per l'usuari a crear.

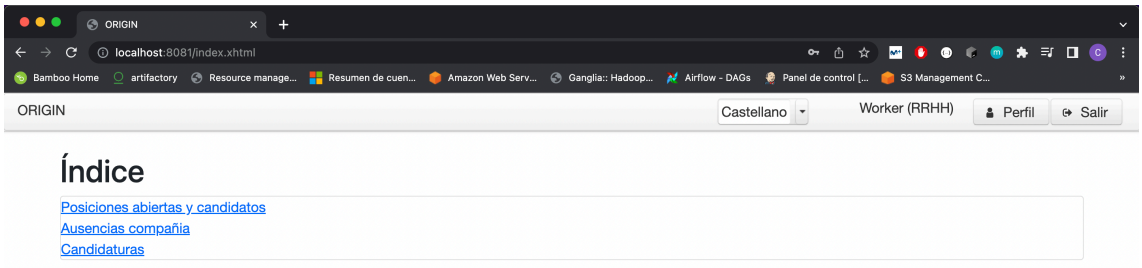
Un cop creats els usuaris llavors apareixen les opcions disponibles que són editar, esborrar o veure el perfil amb tota la informació de l'usuari. A més tindrà disponible l' activació/desactivació de l'usuari a la plataforma.



Il·lustració 80: Pantalla gestió usuaris 2

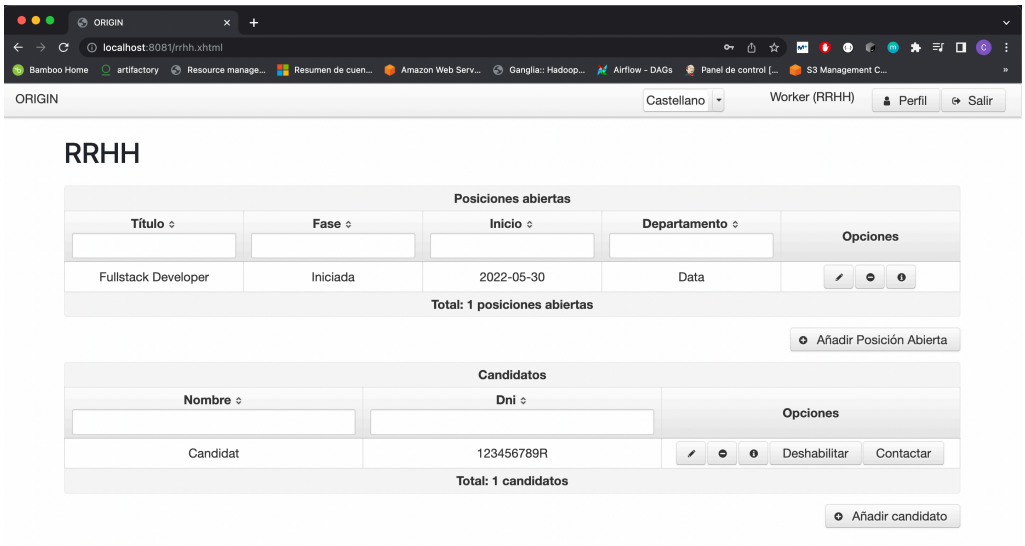
9.3. Recursos Humans

Un cop introduïdes les credencials com a usuari de recursos humans la plataforma redirigeix a l'índex amb les opcions de gestió de posicions obertes, visualització d'absències i visualització de candidatures:



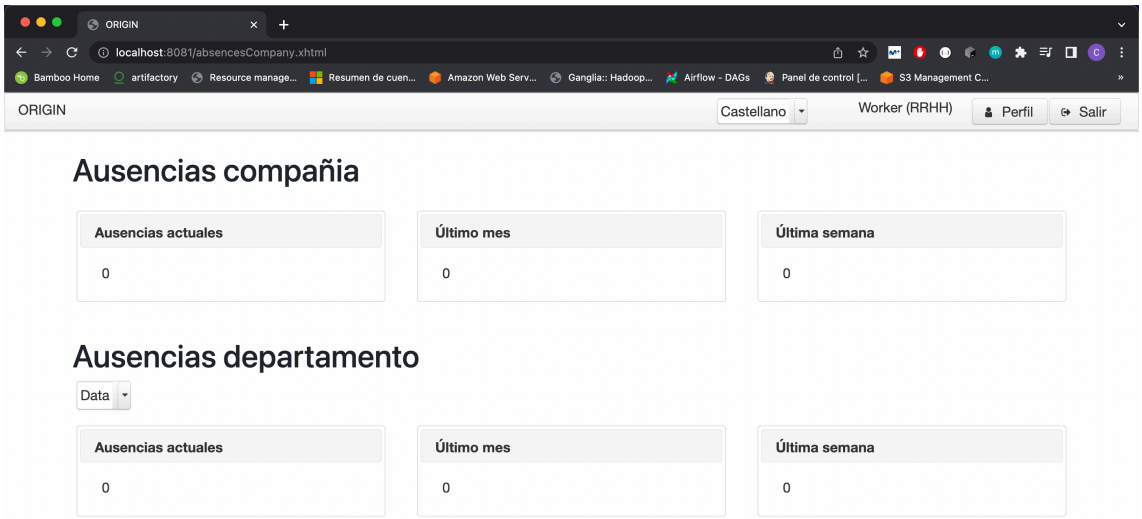
Il·lustració 81: Pantalla índex recursos humans

La primera de les opcions ens mostra dues taules, una per a gestionar les posicions obertes de la companyia amb els respectius filtres i opcions de gestió. L'altra taula proporciona una llista dels candidats inscrits a la plataforma amb les mateixes opcions que la taula anterior a més de la desactivació i l'enviament de missatges per correu electrònic.



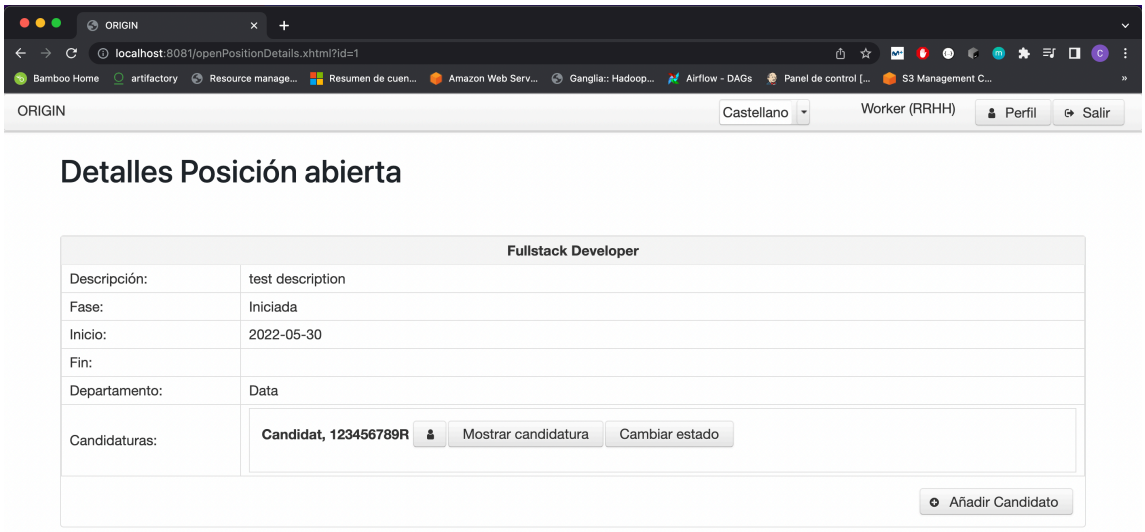
Il·lustració 82: Pantalla gestió posicions obertes i candidats

El segon enllaç pertany a la visualització de les absències en l'àmbit de la companyia com per departament.

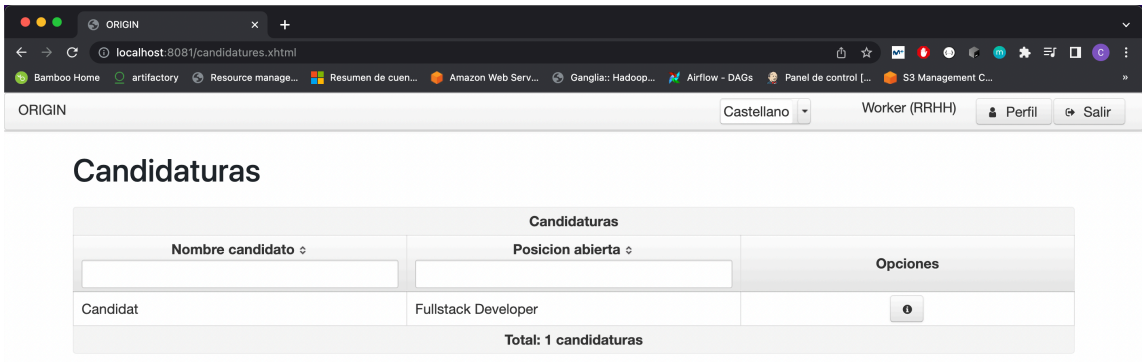


Il·lustració 83: Pantalla visualització absències

L'última de les opcions pertany a la visualització i cerca de les candidatures. En aquest cas apareixerà una candidatura quan afegim un candidat a una posició oberta mitjançant la pantalla d'informació de posició oberta.

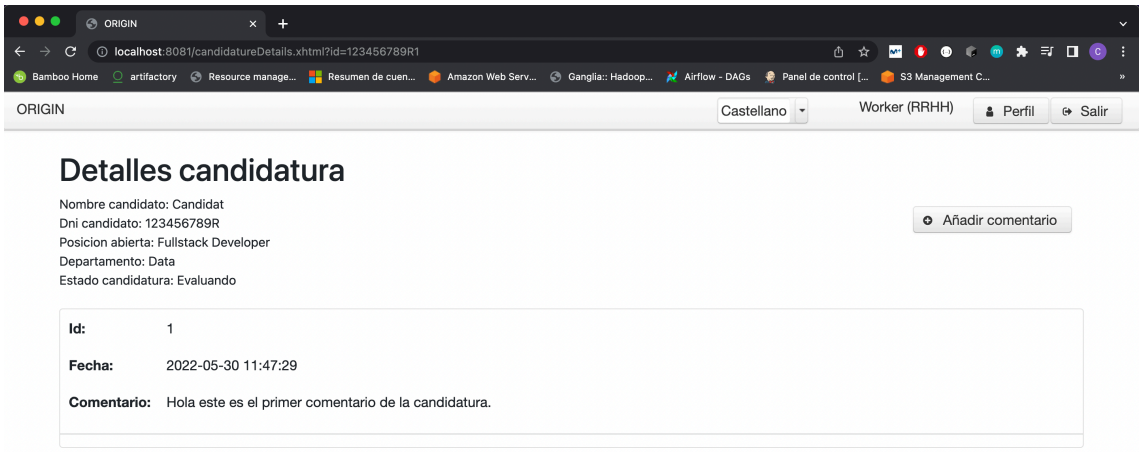


Il·lustració 84: Pantalla detalls posició oberta



Il·lustració 85: Pantalla visualització candidatures

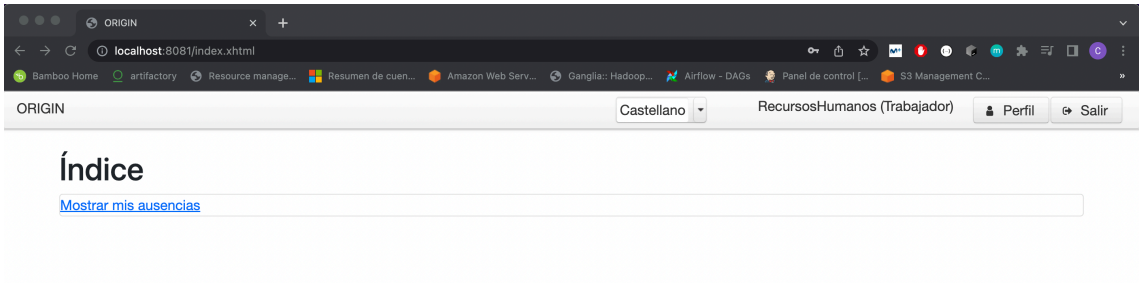
A més aquest usuari tindrà la possibilitat d'afegir comentaris a la pàgina de detall de la candidatura perquè el candidat tingui transparència de l'estat del procés de selecció.



Il·lustració 86: Pantalla detalls candidatura

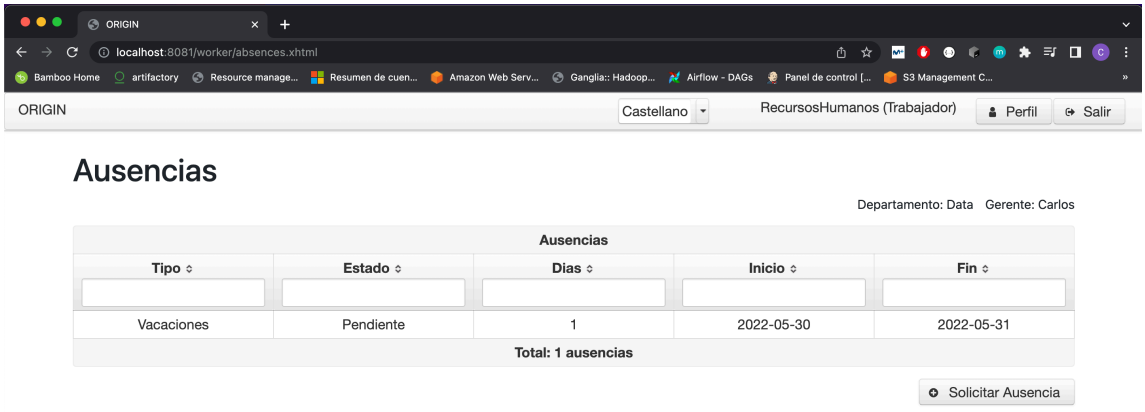
9.4. Treballador

L'autenticació com a treballador farà que la plataforma mostri l'índex amb un enllaç on l'usuari podrà visualitzar i generar absències que hauran de ser aprovades pel seu gerent.



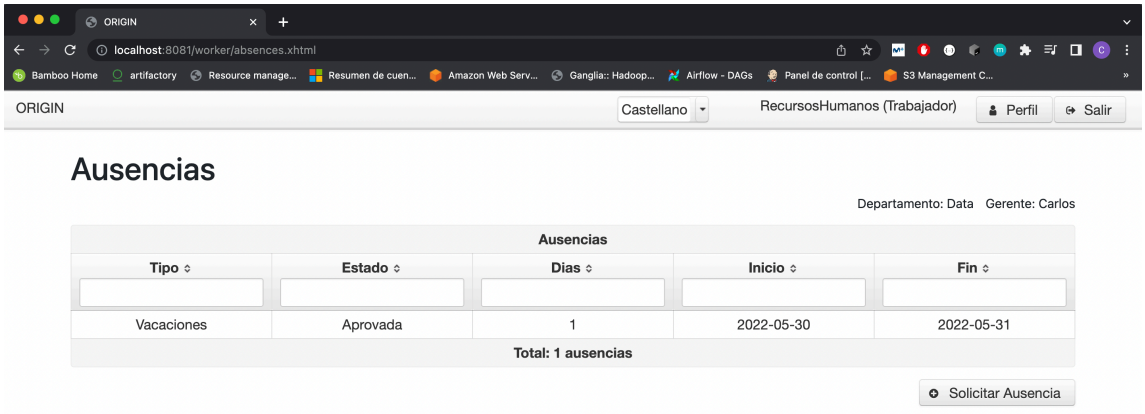
Il·lustració 87: Pantalla índex treballador

A la següent pantalla es mostrarà una taula amb les absències sol·licitades pel treballador. Un cop es generi una absència aquesta apareixerà amb l'estat "Pendiente", ja que necessitarà l'aprovació per part del gerent per canviar d'estat.



Il·lustració 88: Pantalla absències treballador 1

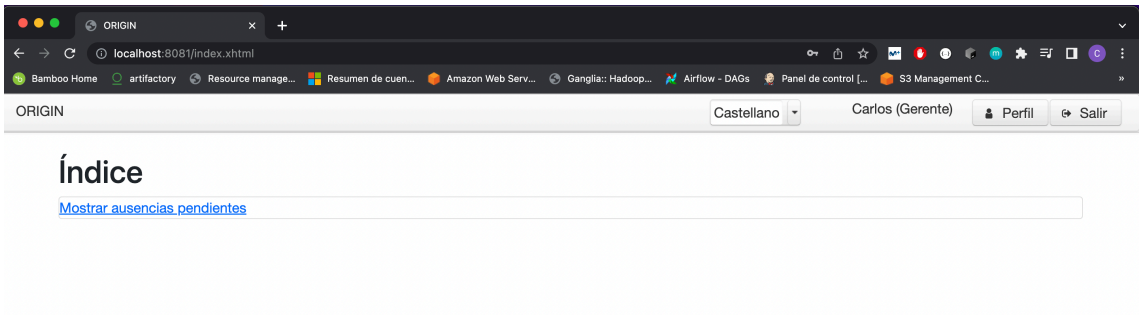
Un cop s'aprovi o es declini l'absència, l'estat d'aquesta es modificarà i es podrà visualitzar a la taula.



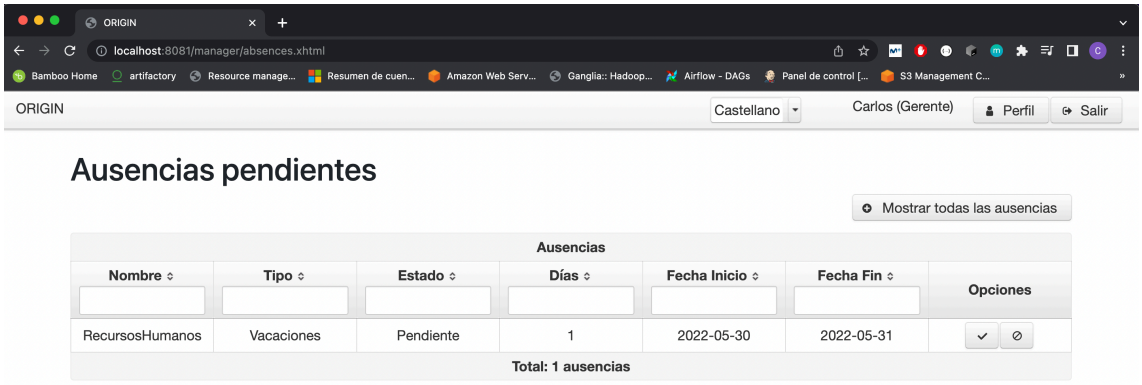
Il·lustració 89: Pantalla absències treballador 2

9.5. Gerent

Un cop autenticat com a gerent, la plataforma ens redirigirà a la pantalla d'índex on trobarem un enllaç per visualitzar aquelles absències que han demanat els treballadors assignats a aquest gerent.



Il·lustració 90: Pantalla índex gerent



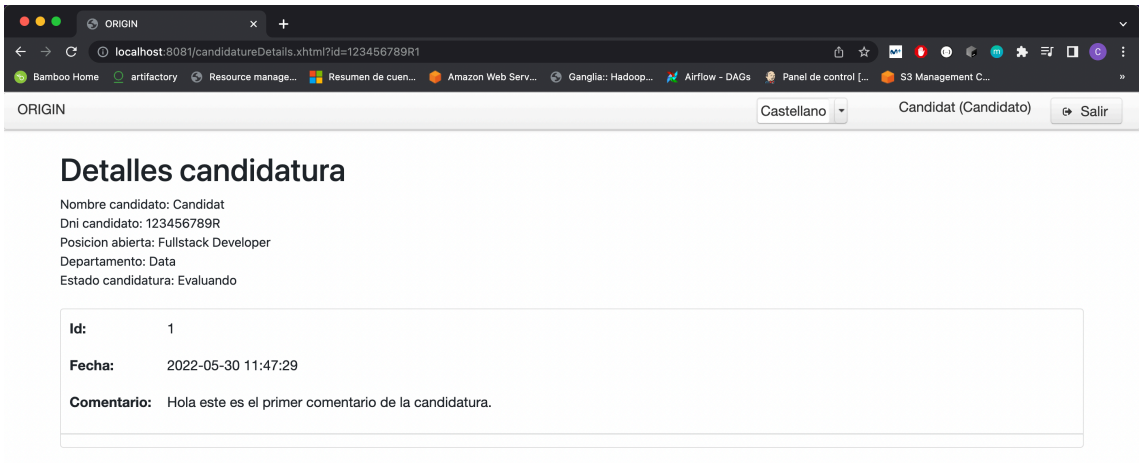
Il·lustració 91: Pantalla absències gerent 1

Un cop aprovada o declinada l'absència, aquesta desapareixerà de la llista, però clicant el botó "Mostrar todas las ausencias" apareixeran totes aquelles que el gerent ja ha gestionat prèviament.

Il·lustració 92: Pantalla absències gerent 2

9.6 Candidat

Com a candidat l'usuari podrà accedir a la plataforma i visualitzar l'estat de les seves candidatures així com comentaris que escriuran els recursos humans en relació amb la candidatura.



Il·lustració 93: Pantalla detalls candidatura candidat

Finalment, cal indicar que si els usuaris tenen inserits correctament els seus correus electrònics llavors, tindran notificacions per correu electrònic de la majoria d'interaccions per optimitzar els accessos a la plataforma.