



Grau en Enginyeria Informàtica

PROJECTE FINAL DE GRAU

Desenvolupament d'una aplicació de gestió de fitxers col·laborativa

Autor:

Nom Cognoms Autor

Tutors:

Dr. Josep Antoni Martí

MEMÒRIA

Convocatòria:

Juny 2025

Departament :
Enginyeria Informàtica

Projecte: Projecte Final de Grau
Document: Memòria
Títol: Desenvolupament d'una aplicació de gestió de fitxers col·laborativa
Autor: Nom Cognoms Autor
Data: Juny 2025

Estudi:
Grau en Enginyeria Informàtica
Universitat de Girona

Supervisor 1:
Dr. Josep Antoni Martí
Universitat de Girona
Email: josep.marti@udg.edu
Web: [Link](#)

Índex

1	Introducció, motivacions, propòsits i objectius	1
1.1	Context del projecte	1
1.2	Motivacions	1
1.3	Propòsit del projecte	2
1.4	Objectius generals	2
1.5	Objectius específics	2
1.6	Situació inicial i públic objectiu	3
2	Estudi de viabilitat	4
2.1	Viabilitat tècnica	4
2.2	Viabilitat econòmica	5
2.3	Viabilitat humana	5
2.4	Viabilitat legal	6
2.5	Conclusió	6
3	Metodologia seguida	7
3.1	Marc teòric: voluntat d'un enfocament àgil	7
3.2	Seguiment i control	7
3.3	Definition of Done: l'únic criteri formal	7
3.4	Fases del projecte: del pla a la realitat	8
3.5	Desglossament i revisió de les unitats de treball	9
3.6	Feedback dels testers	9
3.7	Dificultats de gestió i conciliació	9
3.8	Gestió de riscos i lliçons apreses	9
3.9	Lliçons apreses	9
4	Planificació	10
4.1	Pla original	10
4.2	Execució real i validacions	10
4.3	Pla vs. execució: la breixa quantificada	11
4.4	Causes de les desviacions	13
4.5	Accions correctores aplicades	13
4.6	Estimació d'hores dedicades	13
4.7	Lliçons finals	13
5	Marc de treball i conceptes previs	15
5.1	Introducció	15
5.2	Entorn de desenvolupament	15
5.3	Eines de desenvolupament	15
5.4	Frontend web	17
5.5	Backend i microserveis	22

5.6	Client d'escriptori	27
5.7	Utilitats addicionals	28
5.8	Versionat de dependències	29
6	Requisits del sistema	30
6.1	Introducció	30
6.2	Requisits funcionals	30
6.2.1	1. Gestió d'usuaris	30
6.2.2	2. Gestió d'arxius	30
6.2.3	3. Sistema de paperera	31
6.2.4	4. Compartició d'arxius	31
6.2.5	5. Sincronització en temps real	31
6.2.6	6. Panell d'administració	31
6.3	Requisits no funcionals	31
6.3.1	1. Rendiment	31
6.3.2	2. Seguretat	32
6.3.3	3. Usabilitat	32
6.3.4	4. Mantenibilitat	32
6.3.5	5. Compatibilitat	32
6.3.6	6. Escalabilitat	32
6.3.7	7. Portabilitat	33
6.4	Requisits de maquinari i programari	33
6.5	Resum	33
7	Estudis i decisions	34
7.1	Introducció	34
7.2	Pila Tecnològica del Backend	34
7.2.1	Spring Boot	35
7.2.2	Spring Data JPA	35
7.2.3	PostgreSQL	36
7.2.4	Ecosistema Spring Cloud	36
Spring Cloud Gateway	36	
Eureka	37	
Spring Cloud OpenFeign	38	
7.2.5	Spring Security	38
7.2.6	RabbitMQ	38
7.3	Pila Tecnològica del Frontend Web	39
7.3.1	React	39
7.3.2	React Query (TanStack Query)	40
7.3.3	Zustand	40
7.4	Pila Tecnològica del Client d'Escriptori	40
7.4.1	Tauri i Svelte	40
7.5	Maquinari i infraestructura	41
7.6	Eines de disseny i estil (UI/UX)	42
7.7	Cadena d'eines i DevOps	42
7.8	Llicències i compliment legal	42
7.9	Traçabilitat global amb els requisits	43
8	Anàlisi i disseny del sistema	46
8.1	Introducció	46
8.2	Anàlisi funcional	46

8.2.1	Actors i mòduls	46
8.2.2	Casos d'ús principals	47
8.2.3	Matriu de traçabilitat entre requisits i casos d'ús	50
8.2.4	Diagrama de casos d'ús	52
8.2.5	Diagrames d'activitat dels Casos d'Ús Principals	53
	UC-01: Registrar-se	53
	UC-02: Iniciar sessió	54
	UC-08: Crear o pujar arxius	55
	UC-10: Descarregar	55
	UC-11: Enviar a la paperera	56
	UC-12: Eliminar permanentment	57
	UC-13: Compartir arxius	57
8.3	Arquitectura del sistema	58
8.3.1	Visió general i components	58
8.3.2	Disseny dels microserveis	60
	UserAuthentication	60
	UserManagement	61
	FileManagement	62
	FileAccessControl	63
	FileSharing	64
	Trash	65
	SyncService	65
8.3.3	Patrons de disseny i principis arquitectònics	66
8.3.4	Justificació de l'arquitectura de microserveis	68
8.4	Disseny de les interfícies d'usuari	69
8.4.1	Client web	69
	Autenticació	69
	Escriptori principal	69
8.4.2	Client d'escriptori	76
8.5	Consideracions de seguretat	80
8.5.1	Control d'accés basat en rols (RBAC)	81
8.5.2	Mecanismes de protecció	81
8.5.3	Justificació del compliment del RGPD	82
8.6	Cobertura dels requisits no funcionals	82
8.7	Conclusions	83
9	Implementació i proves	84
9.1	Visió general de la implementació	84
9.2	Entorn de desenvolupament i desplegament	85
9.3	Implementació del backend	86
9.3.1	Gestió d'usuaris (UserAuthentication)	86
9.3.2	Gateway i filtre de JWT	88
9.3.3	Gestió d'arxius (FileManagement)	90
9.3.4	Compartició (FileSharing)	92
9.3.5	Papelera i eliminació asíncrona (TrashService)	94
9.3.6	Servei de sincronització (SyncService)	96
A	Fitxes completes de Casos d'Ús	99
A.1	UC-01: Registrar-se	99
A.2	UC-02: Iniciar sessió	100
A.3	UC-03: Actualitzar perfil	100

A.4	UC-04: Cercar usuaris	101
A.5	UC-05: Llistar usuaris (administració)	101
A.6	UC-06: Actualitzar usuari (administració)	102
A.7	UC-07: Eliminar usuari (administració)	102
A.8	UC-08: Crear o pujar arxius	103
A.9	UC-09A: Renomenar un element	103
A.10	UC-09B: Moure un element	104
A.11	UC-09C: Copiar un element	104
A.12	UC-10: Descarregar	105
A.13	UC-11: Enviar a la paperera	105
A.14	UC-12: Eliminar permanentment	106
A.15	UC-13: Compartir arxius	106
A.16	UC-13A: Revocar accés a un arxiu (propietari)	107
A.17	UC-13B: Deixar de seguir un arxiu compartit (receptor)	107
A.18	UC-14: Actualització en temps real	108
A.19	UC-15: Canviar contrasenya	108
A.20	UC-16: Eliminar compte	109
A.21	UC-17: Restaurar des de la paperera	109
A.22	UC-18: Modificar contrasenya d'un altre usuari (Superadministració)	110
A.23	UC-19: Modificar un administrador (Superadministració)	110
A.24	UC-20: Eliminar un administrador (Superadministració)	111
A.25	UC-21: Modificar nivell de permisos d'un usuari (Superadministració)	111
A.26	UC-22: Sincronitzar arxius compartits	112
B	Diagrams d'activitat de tots els Casos d'Ús	113
	UC-01: Registrar-se	113
	UC-02: Iniciar sessió	114
	UC-03: Actualitzar perfil	114
	UC-04: Cercar usuaris	115
	UC-05: Llistar usuaris (administració)	115
	UC-06: Actualitzar usuari (administració)	116
	UC-07: Eliminar usuari (administració)	116
	UC-08: Crear o pujar arxius	117
	UC-09A: Renomenar un element	118
	UC-09B: Moure un element	118
	UC-09C: Copiar un element	119
	UC-10: Descarregar	119
	UC-11: Enviar a la paperera	120
	UC-12: Eliminar permanentment	121
	UC-13: Compartir arxius	121
	UC-13A: Revocar accés a un arxiu	122
	UC-13B: Deixar de seguir un arxiu	123
	UC-14: Actualització en temps real	123
	UC-15: Canviar contrasenya	124
	UC-16: Eliminar compte	125
	UC-17: Restaurar des de la paperera	125
	UC-18: Modificar contrasenya d'un altre usuari	126
	UC-19: Modificar un administrador	127
	UC-20: Eliminar un administrador	127
	UC-21: Modificar rol d'un usuari	128
	UC-22: Sincronitzar arxius compartits	128

Bibliografia	130
---------------------	------------

Capítol 1

Introducció, motivacions, propòsits i objectius

1.1 Context del projecte

En l'era digital actual, la gestió d'arxius personals i compartits s'ha convertit en una necessitat fonamental tant per a usuaris individuals com per a grups de treball, famílies o petites organitzacions. La creixent dependència de serveis d'emmagatzematge al núvol ha facilitat l'accés remot, la compartició i la sincronització de documents, però també ha generat noves problemàtiques relacionades amb el cost, la privacitat, la dependència tecnològica i la manca de control per part de l'usuari final.

Durant la meva experiència personal amb serveis com Google Drive, OneDrive o Mega, vaig identificar diverses limitacions que em van portar a plantejar aquest projecte. Aquestes solucions, encara que funcionals, imposen restriccions pel que fa a l'espai disponible, requereixen subscripcions per accedir a funcionalitats completes i no ofereixen un control total sobre les dades. Davant d'aquesta situació, vaig decidir desenvolupar una alternativa lliure, extensible i autogestionada.

La solució que proposo es basa en una arquitectura de microserveis, i està dissenyada per ser accessible tant des d'una interfície web com des d'una aplicació d'escriptori multiplataforma. Gràcies a l'ús de contenidors Docker, el desplegament del sistema resulta senzill fins i tot per a usuaris sense coneixements tècnics avançats en oferir també un conjunt d'scripts d'instal·lació automatitzats.

1.2 Motivacions

La meva motivació principal ha estat disposar d'una eina gratuïta, de codi obert i sense restriccions artificials d'ús, que ofereixi una funcionalitat comparable a la dels serveis comercials actuals, retornant el control de les dades a l'usuari final.

Durant el procés de desenvolupament d'aquest projecte, va mancar l'anàlisi de les eines lliures existents, ja que desconeixia la seva existència, un error per part meva que podria haver fet que replantegés el projecte des d'una direcció diferent, però la meva experiència amb serveis comercials va ser suficient per detectar mancances importants pel que fa a accessibilitat, privacitat i control. Aquest projecte neix del

desig de superar aquestes limitacions, desenvolupant una solució que pogués ser utilitzada per qualsevol, sense cap cost i amb plena autonomia sobre les seves dades.

A més, el caràcter obert del projecte permet fomentar la col·laboració, l'aprenentatge i la millora contínua per part de la comunitat, alineant-se amb els principis del programari lliure i la sobirania digital, oferint al final una solució lliure, distribuïda i extensible.

1.3 Propòsit del projecte

El propòsit general d'aquest treball és desenvolupar una plataforma de gestió d'arxius al núvol que permeti la sincronització i compartició de fitxers entre múltiples usuaris, mitjançant una arquitectura distribuïda basada en microserveis. El sistema haurà d'estar preparat per al seu ús tant des d'una aplicació web com des d'una aplicació d'escriptori, adaptant-se així a diferents escenaris i dispositius.

He dissenyat aquesta plataforma amb la intenció que sigui autogestionada, reproducible mitjançant contenidors i accessible per a usuaris sense coneixements tècnics, gràcies a la inclusió de scripts d'instal·lació automatitzats. El meu objectiu és facilitar a qualsevol persona la possibilitat de desplegar el seu propi núvol privat de forma gratuïta, segura i senzilla.

1.4 Objectius generals

Els principals objectius generals que em proposo assolir amb aquest projecte són:

- Dissenyar una arquitectura backend modular, basada en microserveis, que permeti una alta escalabilitat, mantenibilitat i separació de responsabilitats.
- Desenvolupar una interfície web moderna utilitzant React i Tailwind CSS, amb un disseny centrat en l'experiència d'usuari.
- Crear una aplicació d'escriptori multiplataforma utilitzant Tauri i Svelte, que proporcioni accés complet a les funcionalitats del sistema i sigui compatible amb diferents sistemes operatius a més d'utilitzar el mínim de recursos perquè pugui ser utilitzat en dispositius amb poca capacitat.
- Facilitar el desplegament i ús del sistema mitjançant contenidors Docker i scripts d'instal·lació que minimitzin la intervenció tècnica de l'usuari.
- Publicar tot el sistema sota llicència de codi obert (MIT).

1.5 Objectius específics

A més dels objectius generals, he definit una sèrie d'objectius tècnics concrets que permetran donar suport a les funcionalitats requerides per la plataforma:

- Implementar un sistema d'autenticació d'usuaris segur i extensible.
- Desenvolupar microserveis independents per a la gestió de fitxers, compartició entre usuaris, paperera de reciclatge i sincronització d'estats.

- Implementar un sistema de control de permisos que permeti definir els nivells d'accés als arxius compartits.
- Integrar mecanismes de sincronització en temps real mitjançant WebSockets, que assegurin l'actualització immediata de l'estat dels arxius entre tots els clients connectats.
- Incloure una interfície de paperera per a la recuperació d'arxius eliminats de forma accidental.
- Assegurar la portabilitat del sistema entre diferents plataformes mitjançant l'ús de contenidors Docker i fitxers de configuració estàndard (com docker-compose.yml).

1.6 Situació inicial i públic objectiu

Abans de començar el projecte, comptava amb una base sòlida en programació amb Java i Spring Boot, així com en l'ús de contenidors Docker. També tenia coneixements funcionals en desenvolupament web amb React i una comprensió teòrica del paradigma de microserveis, encara que sense experiència pràctica prèvia.

He dissenyat el sistema pensant en petits grups d'usuaris —com famílies o grups d'amics— que desitgin disposar d'una solució al núvol privada i de fàcil instal·lació. Per això, he desenvolupat scripts automatitzats que simplifiquen el procés de desplegament i configuració, eliminant la necessitat de coneixements tècnics avançats.

Finalment, la meva intenció inicial era continuar el desenvolupament del sistema després de la finalització del Treball de Fi de Grau, incorporant noves funcionalitats i millors, i consolidant-lo com una eina lliure, distribuïda i extensible per a la gestió d'arxius personals, però durant el transcurs del desenvolupament del projecte vaig descobrir tota una comunitat de desenvolupadors que estaven treballant en un projecte similar, la qual cosa va fer que em plantegi a futur primer investigar les capacitats dels projectes ja establerts (ja que són més madurs i tenen una comunitat activa al darrere) i finalment decidir si el meu projecte pot aportar una solució millorada o si ja estan cobertes aquestes necessitats.

Capítol 2

Estudi de viabilitat

2.1 Viabilitat tècnica

Per desenvolupar aquest projecte vaig optar per eines i llenguatges amb els quals ja comptava amb experiència prèvia o que oferien clars avantatges tècnics. En el backend vaig utilitzar Java amb Spring Boot a causa de la seva maduresa, extensibilitat i que ja havia treballat amb aquest entorn a nivell professional. Aquesta elecció em va permetre aplicar bones pràctiques de desenvolupament, integrant biblioteques com MapStruct o Spring Cloud per millorar la productivitat i la mantenibilitat del codi.

Per a la interfície web vaig seleccionar React amb Vite i Tailwind CSS. React em va permetre construir una aplicació reactiva, modular i reutilitzable, mentre que Vite va millorar notablement els temps d'arrencada i recàrrega en calent durant el desenvolupament. L'elecció de Tailwind CSS es va justificar per la seva eficiència en la definició d'estils directament des del codi dels components, sense necessitat d'arxius CSS separats.

Pel que fa al client d'escriptori, vaig decidir utilitzar Tauri amb Svelte. Tot i que inicialment vaig considerar reutilitzar la interfície de React, finalment vaig optar per Svelte pel seu menor consum de recursos. Svelte no inclou un runtime, la qual cosa genera binaris més lleugers i amb menor ús de memòria, especialment rellevant en entorns d'escriptori. Aquesta decisió es va basar en la documentació de Tauri i en el consell d'una persona del meu entorn amb una gran quantitat d'experiència en desenvolupament d'aplicacions Tauri a nivell personal.

L'arquitectura es recolza en microserveis, que s'orquesten mitjançant Docker i Docker Compose. El fitxer compose.yml defineix els següents serveis:

- **user-auth**: gestió d'autenticació i tokens.
- **user-management**: gestió de dades d'usuari.
- **file-access**: control d'accés a arxius.
- **file-sharing**: compartició d'arxius entre usuaris.
- **trash**: paperera de reciclatge.
- **file-manager**: pujada, descàrrega i metainformació de fitxers i carpetes.

- **sync**: gestió de sincronització en temps real.
- **gateway**: punt d'entrada unificat per al client web i d'escriptori.
- **eureka**: descobriment de serveis.
- **postgres**: base de dades.
- **rabbitmq**: sistema de missatgeria per a comunicació asíncrona.

Aquesta configuració permet aixecar tota la infraestructura amb una sola comanda, garantint que l'entorn de proves és reproduïble i estàndard en tots els desplegaments, a més de fer l'entorn agnòstic al sistema operatiu que l'ha de córrer, gràcies a les capacitats dels contenidors Docker. La base de dades PostgreSQL es comparteix entre serveis, però cadascun gestiona les seves pròpies taules, evitant dependències creuades. Aquesta va ser una decisió de disseny per seguir les bones pràctiques de disseny de microserveis. RabbitMQ permet una comunicació desacoblada entre serveis que requereixen notificació o processament en segon pla, com la sincronització o el buidatge de la paperera.

L'entorn de desenvolupament es va basar en VSCode com a IDE principal, aprofitant la seva extensió per a Java, el suport per a React i el terminal integrat. Les proves es van realitzar inicialment amb Postman i, més endavant, mitjançant una llista de proves funcionals mantinguda manualment (documentada al fitxer tests.md) per garantir la cobertura de totes les funcionalitats del sistema tant al backend com al frontend.

2.2 Viabilitat econòmica

El cost econòmic del projecte ha estat nul. Totes les eines utilitzades són de codi obert i gratuïtes. Java, React, Vite, Svelte, Tauri, Docker, Git, VSCode i Postman no requereixen llicències de pagament. A més, el projecte s'ha desenvolupat en un ordinador personal ja disponible, equipat amb un processador Intel i7 i 32 GB de RAM, de manera que no ha calgut adquirir maquinari addicional.

Tampoc es preveuen costos de manteniment, ja que les eines emprades són estables, àmpliament documentades i un estàndard de la indústria, fet que n'afavoreix el suport a llarg termini i la possibilitat d'iniciar col·laboradors externs.

2.3 Viabilitat humana

Aquest projecte s'ha desenvolupat de forma individual. Initialment vaig planificar dedicar les tardes i els caps de setmana durant diversos mesos, amb una estimació d'unes 640 hores de treball en total. Tot i que aquesta planificació va resultar massa optimista i no es va seguir de manera exacta, considero que el temps real invertit ha estat raonable i que, si s'hagués mantingut, hauria estat suficient per finalitzar el projecte, com es detalla al Capítol 4.

El projecte també es va concebre com una oportunitat per aprendre noves tecnologies. Es va fixar l'objectiu explícit d'aprofundir en l'ús de React, experimentar amb

arquitectures basades en microserveis, treballar de prop amb tecnologies consolidades com RabbitMQ o Eureka i explorar eines fins aleshores desconegudes com Svelte, Rust i Tauri.

L'organització del treball va incloure l'ús de Git per al control de versions. El codi final està disponible en un repositori dedicat al projecte. El repositori original, utilitzat durant les primeres fases del desenvolupament, es va allotjar en un compte personal vinculat a altres projectes privats; per això es va decidir migrar-lo a un nou repositori exclusiu per a la seva difusió pública.

2.4 Viabilitat legal

Tot i que el projecte s'ha desenvolupat tenint en compte el marc legal vigent (RGPD i LOPDGDD), cal remarcar que es tracta d'una eina de codi obert destinada a ser desplegada i gestionada per tercers. Per tant, la responsabilitat última sobre el compliment de la normativa de protecció de dades recau en l'usuari o organització que decideixi utilitzar la plataforma en un entorn real.

Per facilitar aquest compliment, el projecte inclou plantilles base d'avís legal i política de privacitat (ubicades al directori /legal), que han de ser revisades i adaptades per qui desplegui la solució. Tot i això, en aquesta versió, el sistema no implementa totes les mesures tècniques avançades de seguretat (com el xifratge de dades en trànsit), per la qual cosa es recomana aplicar les mesures addicionals necessàries segons la normativa.

El projecte es publica sota llicència MIT, que en permet l'ús, la modificació i la redistribució sense restriccions, sempre que es conservi l'avís de llicència. La inclusió de les plantilles legals esmentades, juntament amb la llicència permissiva, busca oferir una base sòlida per a un ús responsable i legal de l'eina.

2.5 Conclusió

Considero que el projecte és tècnicament viable gràcies a l'experiència prèvia en les tecnologies seleccionades, l'ús d'eines madures i la modularitat de l'arquitectura. A més, el fet d'utilitzar exclusivament eines de codi obert elimina qualsevol barrera econòmica.

El temps dedicat al projecte, molt superior al previst, posa de manifest que la planificació inicial era massa ambiciosa. La magnitud de la feina va ser més gran de l'esperada, i cal admetre que, fins i tot amb una gestió del temps perfecta, l'abast del projecte probablement superava el que era realista finalitzar en el termini d'un TFG, com es detalla al Capítol 4.

Tot i que el sistema no disposa actualment d'eines de monitoratge ni de logs estructurats, aquest aspecte es contempla com una millora futura que s'abordarà al Capítol 12.

En conjunt, el projecte s'emmarca dins dels recursos disponibles per a un Treball Final de Grau. Les tecnologies emprades i la llicència oberta faciliten que la plataforma sigui adoptada, estesa i millorada fàcilment per altres desenvolupadors o usuaris interessats.

Capítol 3

Metodologia seguida

3.1 Marc teòric: voluntat d'un enfocament àgil

Des de l'inici del projecte, la meva intenció era aplicar una metodologia àgil inspirada en *Scrum*, adaptada a la realitat d'un únic desenvolupador. El plantejament teòric consistia a dividir el treball en sprints curts, amb planificació, desenvolupament i revisió regulars, per tal de poder avançar de manera incremental i adaptar-me als imprevistos.

A la realitat, aquesta estructura àgil va quedar principalment a la fase de planificació. A la pràctica, la temporalitat dels sprints no es va respectar i el desenvolupament es va veure fortament condicionat per la dificultat de compaginar el projecte amb la vida laboral i personal. Aquesta manca de continuïtat i de gestió del temps va ser un dels principals factors que van afectar el progrés, com s'analitza amb més detall al Capítol 4.

3.2 Seguiment i control

En lloc d'utilitzar eines digitals o sistemes formals de seguiment, el control de l'avanç es va fer mitjançant una simple llibreta d'objectius. En aquesta llibreta anotava les tasques a realitzar, afegia nous objectius a mesura que sorgien i anava marcant o tachant els que es completaven. No es va fer servir cap eina de gestió de projectes ni registre digital, fet que en retrospectiva considero un error, ja que va dificultar la visibilitat global del progrés i la prioritització de tasques.

La revisió de les tasques es feia una vegada es donava com a completades, fent un test de la funcionalitat i si es completava es marcava com a completada. Era durant aquest procés que pasava per un període de reflexió sobre el funcionament per a contemplar si la implementació era la correcta o no i si necessitava d'una nova interpretació.

3.3 Definition of Done: l'únic criteri formal

Tot i la manca d'un sistema de seguiment estructurat, sí que vaig mantenir una *Definition of Done* clara per a cada tasca o funcionalitat. Aquest criteri va ser l'únic element formalment aplicat durant tot el projecte i em va ajudar a garantir un mínim de qualitat i coherència en el resultat final.

Definition of Done

Per evitar ambigüïtats vaig definir aquests criteris de tancament:

- El projecte compilava sense errors i la imatge Docker es generava correctament.
- Totes les proves funcionals definides passaven sense incidències.
- La funcionalitat era accessible i usable des de la interfície client corresponent (web o escriptori).
- En cas de ser un dels blocs principals del desenvolupament, es feia una breu sessió exploratòria i cap dels testers troava errors crítics.

Aquesta *Definition of Done* va ser clau per mantenir un estàndard de qualitat constant, tot i la manca d'altres mecanismes de control.

3.4 Fases del projecte: del pla a la realitat

Per estructurar el desenvolupament, inicialment vaig dividir el projecte en vuit fases, tal com es mostra a la Taula 3.1. Aquesta planificació pretenia agrupar els sprints i establir objectius tangibles per a cada etapa.

TAULA 3.1: Fases d'implementació i criteris de finalització

Fase	Criteri de finalització
1. Preparació i aprenentatge	Definició de requisits, entorn Docker operatiu i exploració inicial de Rust/Tauri.
2. Backend inicial	Microserveis d'autenticació, usuaris i gestió d'arxius desenvolupats.
3. Prototip web (MVP)	Client web funcional capaç de pujar i descarregar arxius.
4. Prototip escriptori (Tauri)	Client d'escriptori bàsic amb llistat d'arxius i revisió de l'API.
5. Paperera de reciclatge	Implementació de la funcionalitat de paperera.
6. Compartició i Sync	Incorporació de la compartició d'arxius i sincronització en temps real.
7. Accés a compartits (Tauri)	Client d'escriptori capaç d'accendir a arxius compartits.
8. Admin i desplegament	Panell d'administració creat, proves integrals realitzades i desplegament final.

Aquesta taula reflecteix el disseny original, però la realitat del desenvolupament va ser molt menys lineal. Tal com s'explica al Capítol 4, el projecte va patir desviacions importants respecte al calendari previst: pauses llargues, replanificacions i una execució fragmentada per la dificultat de mantenir la continuïtat. Les fases es van solapar, alguns objectius es van ajornar i d'altres es van modificar o descartar segons la disponibilitat i el feedback rebut.

3.5 Desglossament i revisió de les unitats de treball

Cada fase es traduïa en objectius concrets, però el desglossament i la revisió d'aquestes unitats de treball es feia de manera informal. No hi havia un procés sistemàtic d'anàlisi, desenvolupament i prova dins d'un mateix sprint, sinó que les tasques s'anaven adaptant i reescrivint a mesura que avançava el projecte, sempre anotades a la llibreta.

3.6 Feedback dels testers

El feedback dels testers va ser sempre informal i de paraula, sense cap registre formal. Els comentaris rebuts eren del tipus "m'agrada aquesta part de l'aplicació" o "podries fer que aquesta altra sigui diferent com X". Aquestes aportacions, tot i ser valuoses, no seguien cap procés estructurat ni quedaven documentades més enllà de la meva pròpia memòria o de notes puntuals.

3.7 Dificultats de gestió i conciliació

Un dels principals obstacles va ser la dificultat de compaginar el desenvolupament del projecte amb la vida laboral i personal. Aquesta manca de dedicació continuada, sumada a l'absència d'un sistema de gestió del temps i de seguiment formal, va provocar retards i una execució molt menys àgil del que s'havia previst. Aquesta qüestió s'analitza amb més profunditat al Capítol 4.

3.8 Gestió de riscos i lliçons apreses

Al llarg del projecte vaig identificar diversos riscos, però la manca d'eines de seguiment i la dificultat per mantenir la continuïtat van ser els més rellevants. La taula següent recull els principals riscos i les estratègies de resposta:

TAULA 3.2: Riscos principals i estratègia de mitigació

ID	Risc	Resposta / Mitigació
R1	Falta de temps a causa d'obligacions laborals i personals.	Reprioritzar tasques i posposar característiques no crítiques al pla de treball futur.
R2	Corba d'aprenentatge de Tauri i Rust.	Dedicar una fase específica de formació i buscar ajuda puntual d'un expert.
R3	Absència d'integració contínua, mètriques objectives i eines de seguiment.	Validació manual abans de cada merge; deixar la implantació de CI i eines de gestió com a millora futura.

3.9 Lliçons apreses

Les lliçons apreses apunten clarament a la necessitat d'adoptar eines de seguiment, definir millor la gestió del temps i establir fites intermèdies més rigoroses en futurs projectes. La flexibilitat de l'enfocament àgil va ser útil, però sense disciplina i eines adequades, la planificació es va veure superada per la realitat del dia a dia.

Capítol 4

Planificació

4.1 Pla original

Abans d'escriure la primera línia de codi vaig traçar un calendari ambiciós: acabar el projecte en **vuit mesos** aprofitant tardes i caps de setmana. La idea era avançar de baix a dalt: desenvolupar el backend, validar un prototip web funcional, sumar el client d'escriptori i, finalment, polir i provar tot el conjunt.

TAULA 4.1: Full de ruta inicial (Oct-2023 → Jun-2024)

Mes	Objectiu principal
Oct 2023	Definir requisits, preparar entorn Docker i explorar Rust / Tauri.
Nov-Des 2023	Desenvolupar microserveis d'autenticació, usuaris i operacions d'arxius.
Gen 2024	Construir un prototip web capaç de pujar i descarregar arxius.
Feb 2024	Prototipar el client Tauri i revisar l'API.
Mar 2024	Afegir la paperera.
Abr 2024	Incorporar la compartició d'arxius i la sincronització en temps real.
Mai 2024	Permetre que el client Tauri accedeixi als arxius compartits.
Jun 2024	Crear el panell d'administració, executar proves integrals i desplegar.

4.2 Execució real i validacions

La trajectòria real del projecte va ser força diferent del que s'havia planejat inicialment. El desenvolupament es va allargar fins al juliol de 2025, amb diversos períodes de pausa entremig. Durant aquest temps, vaig realitzar tres validacions principals amb usuaris beta testers: la primera després d'implementar l'administració de fitxers, la segona quan vaig completar la paperera, i l'última un cop finalitzada la sincronització.

Cal destacar que, després de la implementació de la compartició d'arxius, es va detectar un error d'arquitectura en la gestió dels serveis del backend. Aquest error feia que tant el servei de paperera com el de compartició depenguessin del servei de file manager com a punt de contacte, trencant així el principi d'acoblament feble (loose coupling) dels microserveis, que estableix que cada servei ha de ser independent i autònom. Això va obligar a fer un refactor important del backend, que va durar

aproximadament dues setmanes i mitja i es va solapar amb el desenvolupament del servei de compartició. Aquest refactor va requerir tornar a testejar tota l'aplicació per assegurar que no s'havien trencat funcionalitats ja implementades, i va endarrerir l'inici del desenvolupament de la sincronització, que es va començar just després. El detall tècnic d'aquest error i la solució adoptada es tracta amb més profunditat al capítol 8.

Tot això es pot veure reflectit al diagrama de Gantt de la Figura 4.1, on es mostren les diferents etapes del desenvolupament, els períodes d'aturada, les proves amb usuaris (en groc), les millores aplicades (en morat) i el període de refactor (en marró).

Finalment, cal destacar que una de les funcionalitats planificades, la visualització d'arxius compartits des del client d'escriptori, es va haver de descartar a causa de la complexitat tècnica i la falta de temps. Aquesta última etapa de desenvolupament del prototip de Tauri, a més, es va haver de realitzar en paral·lel amb la creació del panell d'administració per tal de complir amb els terminis.

Els beta-tests van durar entre un i dos dies i van ser decisius per polir usabilitat. Gràcies a ells vaig afegir la compartició múltiple d'arxius, accessos ràpids de teclat i components *Tremor Raw*, entre altres millores.

4.3 Pla vs. execució: la bretxa quantificada

TAULA 4.2: Retards acumulats respecte al pla inicial

Fita completada	Pla	Real	Retard / comentari
Auth + Usuaris	Nov-Des 2023	Feb 2024	+2 mesos
Admin de fitxers (MVP)	Feb 2024	Mar 2024	+1 mes; optimització React + accessibilitat
Paperera	Abr 2024	Oct 2024	+6 mesos; pauses intermitents
Compartició	Mai 2024	Nov 2024	+6 mesos; validació + refactor
Accés a compartits (Tauri)	Jun 2024	No implementat	Descartat per complexitat i falta de temps
Refactor backend	—	Nov 2024	2,5 setmanes; revalidació completa
Sync	Mai 2024	Feb 2025	+9 mesos; inici després del refactor
Prototip Tauri	Mar 2024	Jun 2025	+15 mesos; desenvolupament en paral·lel amb panell d'admin
Admin + proves globals	Jun 2024	Jun 2025	+12 mesos
Redacció memòria	—	Jun-Jul 2025	no previst; 2 setmanes exclusives



FIGURA 4.1: Cronograma real amb desenvolupaments, pauses, refactorització i validacions (Oct-2023 → Jul-2025)

4.4 Causes de les desviacions

1. **Discontinuïtat en la dedicació.** Les pauses prolongades van exigir un "peatge cognitiu" que estimo entre un 10 i un 15 % del temps total: reprendre context, re-configurar l'entorn i refrescar la lògica del codi.
2. **Corba d'aprenentatge subestimada.** Rust i Svelte van requerir més pràctica de la prevista; la integració amb Tauri es va desplaçar gairebé un any.
3. **Obligacions laborals i viatges.** Entre gener i octubre de 2024 vaig alternar desplaçaments a Granada i Alemanya, fragmentant els cicles de treball.

4.5 Accions correctores aplicades

- Vaig re-prioritzar el backlog: vaig desplaçar el prototip de Tauri al tram final, sacrificant la funcionalitat de visualitzar arxius compartits des d'aquest, i vaig deixar la sincronització de fitxers compartits a local per a futures versions.
- Vaig reservar un sprint-buffer al desembre 2024 per absorbir retards acumulats i estabilitzar el full de ruta.

4.6 Estimació d'hores dedicades

TAULA 4.3: Pla vs. realitat en esforç mensual aproximat

Mètrica	Pla	Real
Mitjana mensual	60 h	20 – 40 h
Mes pic	60 h	~45 h
Mes vall	60 h	0 h
Pèrdua per rerees	—	10 – 15 % del temps total

4.7 Lliçons finals

Aquesta experiència em va ensenyar que la **continuïtat** pesa tant com la quantitat d'hores: cada paua llarga afegeix un peatge cognitiu que penalitza el calendari. Igualment, sense **mètriques objectives** (Kanban, full d'hores, CI amb indicadors) és fàcil sobreestimar els avenços.

Per a futurs projectes aplicaré:

1. Un tauler Kanban públic.
2. Buffers explícits després de viatges o pics laborals que redueixin la incertesa.
3. Prototips inicials per validar corbes d'aprenentatge abans de planificar dates ambicioses.
4. Sessions de feedback programades des del primer dia, amb temps reservat per implementar millors.

En definitiva, tot i que els terminis inicials es van dilatar, el cronograma real mostra una evolució honesta, validada amb usuaris i enriquida amb lliçons que ja aplico en la meva pràctica professional.

Capítol 5

Marc de treball i conceptes previs

5.1 Introducció

Aquest Treball de Fi de Grau s'ha desenvolupat de manera independent, sense el suport d'una organització externa, amb l'objectiu d'aprofundir en la construcció de sistemes distribuïts moderns, així com un repte tècnic personal en afrontar un projecte de gran magnitud i complexitat amb múltiples llenguatges i tecnologies. El seguiment acadèmic ha anat a càrrec del professor **Josep Soler** (tutor del projecte).

El desenvolupament del client d'escriptori es va realitzar en col·laboració molt estreta amb **Jawad Adbellaoui**, qui va aportar la seva experiència en Tauri i Rust. Aquesta col·laboració va ser clau per assolir un resultat funcional i em va permetre aprendre de primera mà bones pràctiques en tecnologies emergents.

Al llarg del capítol es descriuen les eines i tecnologies que conformen la pila de desenvolupament. Cada una es presenta especificant el seu propòsit, la motivació de la seva elecció i el seu paper dins l'arquitectura global. Totes elles són de codi obert i d'ús gratuït, fet que reforça la sostenibilitat econòmica discutida al Capítol 2.

5.2 Entorn de desenvolupament

El treball s'ha dut a terme principalment en un portàtil amb **Ubuntu 22.04 LTS** (kernel 5.15) i l'editor **Visual Studio Code** com a IDE principal. La Taula 5.1 —inclosa al final d'aquest capítol— detalla les versions exactes dels llenguatges, marcs i eines utilitzades. Per a les proves inicials de l'API es va utilitzar **Postman**, tot i que el flux de validació va passar ràpidament a la pròpia aplicació web a mesura que el frontend va guanyar funcionalitat.

5.3 Eines de desenvolupament

Sistema operatiu

Ubuntu 22.04 LTS proporciona un entorn estable, actualitzat i àmpliament documentat. El seu gestor de paquets *apt* i la compatibilitat amb contenidors docker-ce van facilitar la instal·lació de dependències i la creació d'entorns reproduïbles.

Git i GitHub

Git és un sistema de control de versions distribuït creat per Linus Torvalds el 2005. La seva arquitectura descentralitzada permet treballar sense connexió, conservar l'històric complet a cada clon i afavorir fluxos paral·lels mitjançant branques lleugeres. GitHub afegeix a Git funcionalitats col·laboratives (*pull requests*, revisions de codi i GitHub Actions) i ha servit d'eix per a la integració contínua i el desplegament automatitzat del projecte.

Visual Studio Code

Visual Studio Code (VS Code) és un editor multiplataforma, lleuger i extensible. Amb extensions com *Rust Analyzer*, *Spring Tools* i suport integrat per a TypeScript, ha permès una experiència de desenvolupament unificada per al *backend* en Java i els clients en React i Tauri/Svelte.

Node.js

Node.js és un entorn d'execució que permet fer servir JavaScript fora del navegador, principalment al servidor. Gràcies a Node.js, es poden crear aplicacions i scripts que s'executen directament a la màquina, com ara eines de desenvolupament, servidors web o processos d'automatització. En aquest projecte, Node.js s'utilitza per executar scripts que ajuden a preparar l'entorn i per donar suport a la construcció i execució del frontend web.

pnpm

pnpm (performant npm) és un gestor de paquets per a Node.js, dissenyat per ser ràpid i eficient en l'ús de l'espai en disc. A diferència de npm, que pot duplicar paquets, pnpm utilitza un magatzem de contingut adreçable on només es desa una versió de cada paquet. Després, crea enllaços simbòlics (symlinks) a la carpeta node_modules del projecte. Aquest enfocament no només redueix dràsticament l'espai en disc necessari, sinó que també accelera significativament els temps d'installació. En aquest projecte, s'ha estandarditzat l'ús de pnpm per gestionar totes les dependències dels frontends (@ui-new i @desktop) i per executar scripts útils durant el desenvolupament i la construcció de les aplicacions, garantint consistència i eficiència.

Postman

Postman és una eina gràfica molt utilitzada per desenvolupar, provar i documentar APIs REST. Permet enviar peticions HTTP (GET, POST, PUT, DELETE, etc.) a un servidor, veure les respostes, gestionar col·leccions de peticions i validar el comportament dels endpoints de manera ràpida i visual. Això facilita la detecció d'errors, la comprovació de contractes i la col·laboració entre equips de backend i frontend.

En aquest projecte, Postman es va utilitzar durant les primeres iteracions per dissenyar i provar les peticions a l'API REST, assegurant que els endpoints funcionaven correctament i que les respostes complien l'estructura esperada. A mesura que el frontend va madurar i va guanyar funcionalitat, les proves es van traslladar a la pròpia interfície web, reduint la dependència de Postman i agilitzant el flux de validació.

Scripts d'instal·lació

Els scripts `setup.sh` (Bash) i `setup.ps1` (PowerShell) automatitzen la preparació de l'entorn: descàrrega de dependències, construcció de contenidors Docker i configuració de variables. Així es garanteix la reproduïibilitat de l'entorn amb una sola comanda.

5.4 Frontend web

React

React (2013) és una biblioteca JavaScript de codi obert creada per Facebook per facilitar la construcció d'interfícies d'usuari (UI) complexes de manera eficient, escalable i mantenible. El seu model es basa en la composició de components: unitats independents i reutilitzables que encapsulen tant la lògica com la presentació, afavorint la reutilització, la testabilitat i la separació de responsabilitats.

Una de les innovacions clau de React és el *Virtual DOM*, una representació en memòria de l'estructura del DOM real. Quan l'estat d'un component canvia, React genera un nou arbre virtual i calcula la diferència respecte a l'anterior (*diffing algorithm*). Només les parts modificades s'actualitzen al DOM real mitjançant el procés de *reconciliació*, minimitzant operacions costoses i millorant el rendiment, especialment en aplicacions dinàmiques o amb moltes actualitzacions.

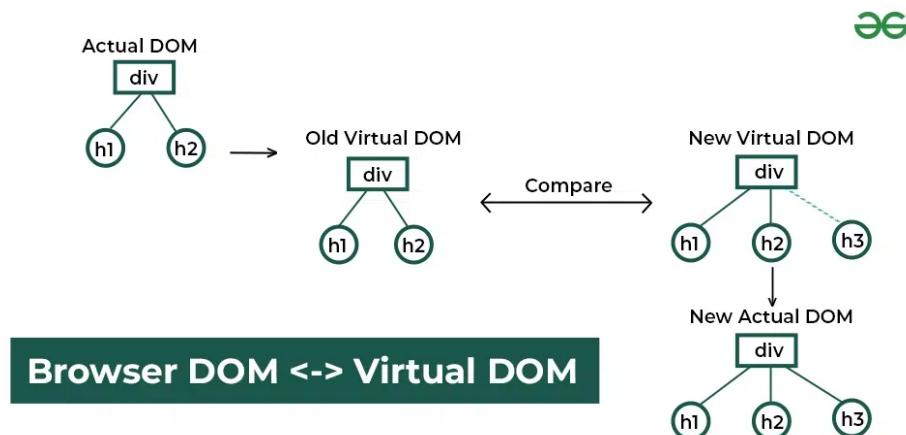


FIGURA 5.1: Procés de reconciliació a React: arbre virtual → diff → pegat sobre el DOM.

El flux de dades en React és unidireccional (de pares a fills), cosa que simplifica el raonament sobre l'estat i redueix la probabilitat d'efectes col-laterals. Les dades es passen als components fills mitjançant *props*, mentre que l'estat local es gestiona dins de cada component. Per compartir estat entre components no relacionats jeràrquicament, es pot utilitzar el context de React o gestors d'estat externs com *Zustand*.

Amb la introducció dels *hooks*, la gestió de l'estat i dels efectes col-laterals es pot fer de manera funcional, sense recórrer a classes. Hooks com `useState`, `useEffect` o

useContext permeten encapsular lògica reutilitzable i millorar la llegibilitat i mantenibilitat del codi.

React destaca també per la seva integració amb l'ecosistema JavaScript modern i una comunitat molt activa. Llibreries com TanStack React Query, Zustand, Radix UI o Vite cobreixen des de la gestió eficient de dades asíncrones fins a l'accessibilitat i l'optimització del frontend. Això permet abordar projectes professionals de qualsevol escala, des de prototips fins a sistemes grans i robustos.

A més, React facilita paradigmes avançats com la renderització al servidor (SSR), les aplicacions d'una sola pàgina (SPA) i la integració amb WebSockets per a temps real. La seva filosofia de considerar la UI com una funció de l'estat ($UI = f(state)$) simplifica la gestió de la complexitat i afavoreix la previsibilitat del comportament de l'aplicació. Tot plegat fa de React una eina idònia per a projectes que requereixen manteniment, escalabilitat i un alt grau d'interactivitat.

TypeScript

TypeScript (2012) amplia JavaScript amb tipatge estàtic i característiques orientades a objectes. El *transpiler* tsc prevé errors en temps de compilació i facilita l'autocompletat, millorant la robustesa entre client i servidor.

Vite

Vite és una eina que ajuda a posar en marxa i construir aplicacions web de manera molt ràpida i senzilla. Durant el desenvolupament, permet veure els canvis a l'instant cada vegada que es modifica el codi, sense haver d'esperar llargues càrregues. Això fa que programar sigui molt més àgil i còmode. Quan arriba el moment de publicar l'aplicació, Vite s'encarrega d'ordenar i optimitzar tots els fitxers perquè la web es carregui ràpidament als usuaris. En aquest projecte, Vite ha estat fonamental per poder desenvolupar i provar la interfície web de forma eficient i sense complicacions.

Tailwind CSS

Tailwind CSS és un framework de CSS utilitari que permet construir interfícies d'usuari modernes i responsives mitjançant l'ús de classes predefinides molt específiques (com p-4, text-lg, bg-blue-500, etc.). A diferència dels frameworks tradicionals basats en components o temes, Tailwind apostava per una filosofia "utility-first", on cada classe aplica un sol estil CSS, la qual cosa permet escriure el disseny directament a l'HTML o JSX de cada component.

Aquesta aproximació té diversos avantatges: elimina la necessitat d'escriure fulls d'estil personalitzats per a cada component, redueix la duplicació de codi i facilita la coherència visual a tota l'aplicació. A més, Tailwind facilita la creació de dissenys responsius i adaptatius gràcies a la seva sintaxi per a punts de trencament i variants d'estat (hover, focus, dark mode, etc.).

Un dels punts forts de Tailwind és el seu procés de "purge" o depuració: durant la construcció de l'aplicació, Tailwind analitza tot el codi font i elimina automàticament les classes que no s'utilitzen, generant un fitxer CSS final molt lleuger i optimitzat

per a producció. Això millora el rendiment de càrrega i l'experiència d'usuari, especialment en aplicacions grans.

En aquest projecte, Tailwind CSS ha permès desenvolupar una interfície moderna, coherent i altament personalitzable, accelerant el procés de maquetació i assegurant una bona experiència visual.

Radix UI

Radix UI és una llibreria de components d'interfície d'usuari per a React, centrada en l'accessibilitat i la composició. A diferència d'altres biblioteques, Radix UI proporciona components "headless", és a dir, sense estils visuals predefinits, de manera que el desenvolupador pot aplicar el seu propi disseny sense perdre funcionalitat ni accessibilitat. Tots els components compleixen les directrius ARIA i gestionen automàticament focus, navegació per teclat i altres requisits d'accessibilitat, la qual cosa garanteix que l'aplicació sigui usable per a tothom. A més, Radix UI facilita la creació de diàlegs, menús, popovers i altres elements complexos sense haver de preocupar-se pels detalls interns de gestió d'estat o focus. En aquest projecte, Radix UI s'ha utilitzat per implementar diàlegs i menús accessibles, assegurant una experiència d'usuari coherent i inclusiva.

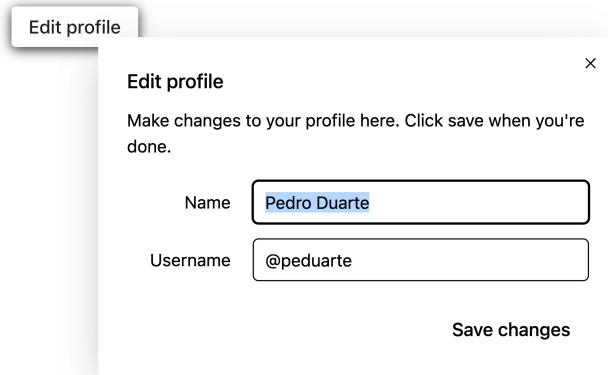


FIGURA 5.2: Diàleg basat en Radix amb focus i navegació accessibles.

React Query

React Query (actualment TanStack Query) és una llibreria per a la gestió eficient de dades asíncrones en aplicacions React. La seva funció principal és facilitar la consulta, la memòria cache i la sincronització de dades provinents d'APIs o serveis externs. React Query permet definir "queries" declaratives que s'executen automàticament quan el component es renderitza, gestionant l'estat de càrrega, error i dades sense necessitat de codi addicional. A més, manté una memòria cache intel·ligent que evita peticions innecessàries i actualitza automàticament les dades quan detecta canvis, aplicant estratègies com "stale-while-revalidate". Això simplifica molt la gestió de dades remotes, millora el rendiment i redueix la complexitat del codi, especialment en aplicacions amb moltes consultes o dades en temps real. En aquest projecte, React Query ha estat clau per mantenir la UI sincronitzada amb el backend i gestionar l'estat de les peticions de manera robusta i eficient.

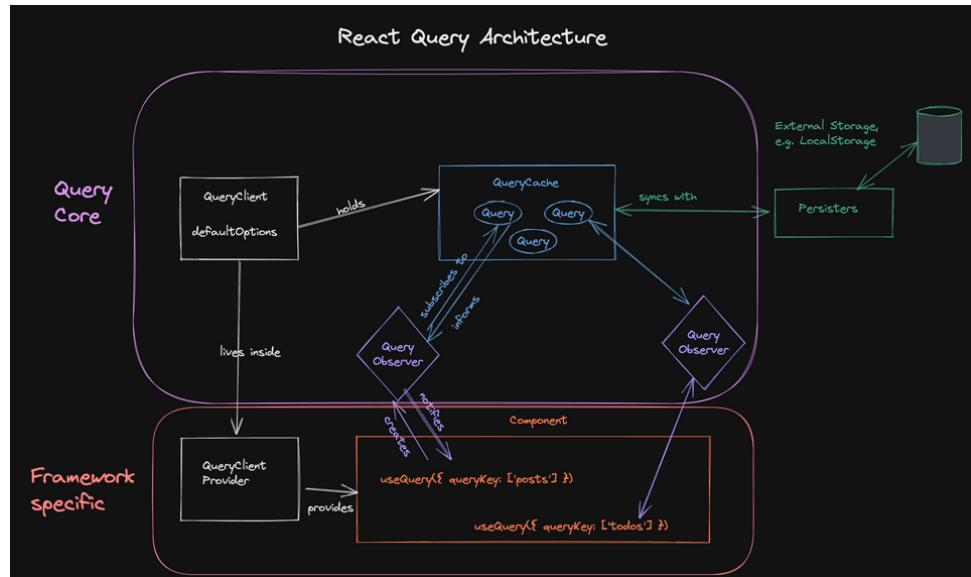


FIGURA 5.3: Circuit de dades a React Query: memòria cache → fetch → revalidate.

El funcionament intern de React Query es pot entendre millor amb l’ajuda de la imatge anterior. Al centre del sistema hi ha el **QueryClient**, que gestiona totes les opcions i la configuració global. Aquest client conté una **QueryCache**, que és la memòria cache on es desen totes les dades de les consultes (queries) realitzades a l’API o backend.

Quan un component React utilitza el hook `useQuery`, crea un **Query Observer** que s’encarrega de subscriure’s a una consulta concreta dins la memòria cache. Si la dada ja existeix a la cache, el component la rep immediatament; si no, React Query fa la petició a l’API i desa la resposta a la cache per a futures consultes. Això permet que diversos components puguin compartir i reutilitzar dades sense repetir peticions innecessàries.

A més, la memòria cache es pot sincronitzar amb un emmagatzematge extern (com LocalStorage) mitjançant els **Persistors**, de manera que les dades es conserven encara que l’usuari recarregui la pàgina. Tot aquest sistema garanteix que l’estat de les dades a la UI estigui sempre actualitzat, gestionant automàticament la recàrrega, la invalidació i la revalidació de la informació segons sigui necessari.

Aquesta arquitectura fa que React Query sigui molt eficient i robust per a la gestió de dades asíncrones en aplicacions React modernes.

Zustand

Zustand és una llibreria lleugera per a la gestió d'estat global en aplicacions React. A diferència de solucions més pesades com Redux, ofereix una API molt minimalista basada en *hooks*: cada *store* és, de fet, un hook creat amb `create()`. Els components consumeixen l'estat cridant aquest hook i poden subscriure's només a la porció que necessiten, cosa que evita re-renderitzacions innecessàries i millora el rendiment. Internament, Zustand utilitza *proxies* per detectar canvis i notificar únicament els

components afectats, facilita la divisió en *slices* per escalar grans projectes i disposa de *middleware* per persistir, registrar o desfer canvis.

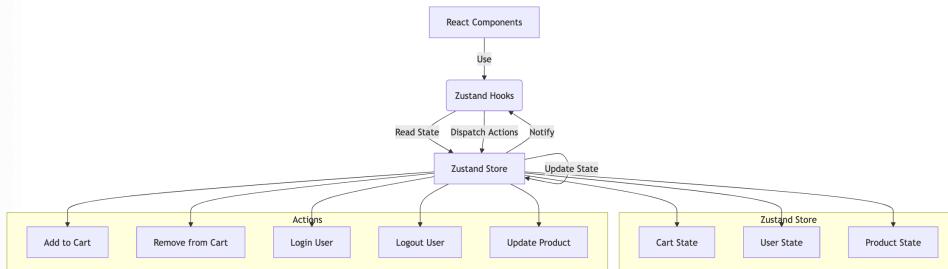


FIGURA 5.4: Diagrama de flux de Zustand en una aplicació React: els components llegeixen o actualitzen l'estat a través dels *hooks*, que deleguen l'operació al *store*; quan aquest actualitza l'estat, només notifica els components subscriptos.

Com es veu al diagrama 5.4, el flux és extremadament directe:

1. **Components React** criden el *hook* del *store* per llegir l'estat o executar accions.
2. El *hook* retorna l'estat sol·licitat i exposa les accions definides dins el *store*.
3. Les **accions** invocades fan `set()` o `get()` per modificar l'estat.
4. El *store* actualitza l'estat i notifica només els components subscriptos al tros de dades canviat, provocant el re-render corresponent.

Aquesta simplicitat permet separar la lògica d'estat dels components, millorar la testabilitat i mantenir el codi net.

En el projecte actual, Zustand s'utilitza per gestionar l'estat global relacionat amb la navegació d'arxius, la selecció múltiple i el context de visualització. Aquesta aproximació permet mantenir la UI sincronitzada i reactiva, simplificant la gestió de l'estat entre components i facilitant la implementació de funcionalitats avançades com la selecció múltiple, el porta-retalls i la navegació entre seccions.

@dnd-kit i Selecto

@dnd-kit és una llibreria modular que abstrau la complexitat de les interaccions d'arrassegars i deixar anar (drag & drop) mitjançant sensors que detecten esdeveniments del ratolí, tàctils i de teclat. La seva arquitectura basada en contexts i proveïdors permet implementar funcionalitats avançades com l'ordenació de llistes o el moviment entre contenidors de forma declarativa.

Per altra banda, Selecto ofereix una API per implementar selecció múltiple mitjançant un rectangle o llaç de selecció, similar al que trobem en aplicacions d'escriptori. Permet configurar la detecció de col·lisions, filtrar elements seleccionables i personalitzar l'aparença visual del llaç.

La combinació d'aquestes dues llibreries ha permès implementar una experiència d'usuari molt similar a la d'un explorador d'arxius natiu, on es poden seleccionar

múltiples elements arrossegant el ratolí i després moure'ls a altres carpetes mitjançant drag & drop, tot mantenint una implementació neta i mantenible.

WebSocket API

L'aplicació utilitza WebSocket, un protocol de comunicació que estableix una connexió bidireccional persistent entre el client web React i el servidor. A diferència del protocol HTTP tradicional, on el client ha de fer peticions explícites per obtenir dades, WebSocket manté un canal obert que permet al servidor enviar informació al client en qualsevol moment. Aquesta característica és especialment útil en la nostra interfície web per implementar funcionalitats en temps real, com la sincronització automàtica quan altres usuaris modifiquen arxius o la recepció instantània de notificacions del sistema.

5.5 Backend i microserveis

Java 17

Java és un llenguatge de programació orientat a objectes àmpliament utilitzat en el desenvolupament d'aplicacions empresarials, sistemes distribuïts i serveis web. La seva principal característica és la portabilitat: el codi Java es compila a bytecode, que pot ser executat en qualsevol plataforma que disposi d'una màquina virtual Java (JVM), independentment del sistema operatiu o el maquinari. Això facilita la creació d'aplicacions escalables i mantenibles, ja que el mateix codi pot desplegar-se en entorns molt diversos sense modificacions. Java ofereix un sistema de tipus fort, gestió automàtica de memòria mitjançant recollida d'escombraries (garbage collection) i una àmplia biblioteca estàndard per a operacions de xarxa, persistència, concorrència i seguretat. A més, la comunitat Java disposa d'un ecosistema ric en frameworks i eines que acceleren el desenvolupament i garanteixen la robustesa de les aplicacions. En el context d'aquest projecte, Java s'utilitza com a llenguatge principal per implementar els microserveis del backend, aprofitant la seva maduresa, estabilitat i suport a llarg termini.

Spring Boot

Spring Boot és un framework que simplifica la creació d'aplicacions Java basades en Spring, especialment microserveis. Proporciona una configuració automàtica intel·ligent i una estructura d'aplicació predefinida, permetent als desenvolupadors centrar-se en la lògica de negoci en lloc de la configuració manual. Un dels seus avantatges clau és la capacitat de generar microserveis autosuficients, que inclouen un servidor Tomcat embedut, eliminant la necessitat de desplegar arxius WAR en servidors externs. Els *starters* de Spring Boot encapsulen conjunts de dependències habituals, facilitant la integració de tecnologies com bases de dades, seguretat, missatgeria o REST. A més, ofereix eines per monitoritzar, provar i desplegar serveis de manera eficient. En aquest projecte, Spring Boot serveix de base per a cada microservei, assegurant una inicialització ràpida, una gestió coherent de dependències i una arquitectura modular i escalable.

Spring Data JPA

Spring Data JPA és un framework que facilita la persistència de dades en bases de dades relacionals mitjançant el patró ORM (Object-Relational Mapping). Permet definir entitats Java que es corresponen amb taules de la base de dades i proporciona una API per realitzar operacions CRUD (crear, llegir, actualitzar, eliminar) sense escriure SQL manualment. Una de les seves funcionalitats més potents és la generació automàtica de consultes JPQL a partir de la nomenclatura dels mètodes (per exemple, `findByUsernameAndStatus`), la qual cosa accelera el desenvolupament i redueix errors. A més, integra mecanismes per a la gestió de transaccions, la paginació, la ordenació i la validació d'entitats. En el projecte, Spring Data JPA s'utilitza per accedir i manipular les dades de cada microservei, assegurant una persistència robusta, coherent i fàcilment testeable.

Eureka

Eureka és el servei de descobriment que manté un registre viu i coherent de tots els microserveis desplegats. Cada instància, tan bon punt s'aixeca, es registra automàticament al servidor i, mitjançant petits *heartbeats* periòdics, confirma que continua disponible. Així, qualsevol altre servei —o bé el Gateway— pot consultar el registre per obtenir l'adreça més recent i balancejar les peticions sense codificar IPs fixes. Aquesta capacitat d'autodescobriment garanteix resiliència i escalabilitat en entorns on les instàncies poden aparèixer o desaparèixer dinàmicament durant pics de càrrega, actualitzacions o fallades.

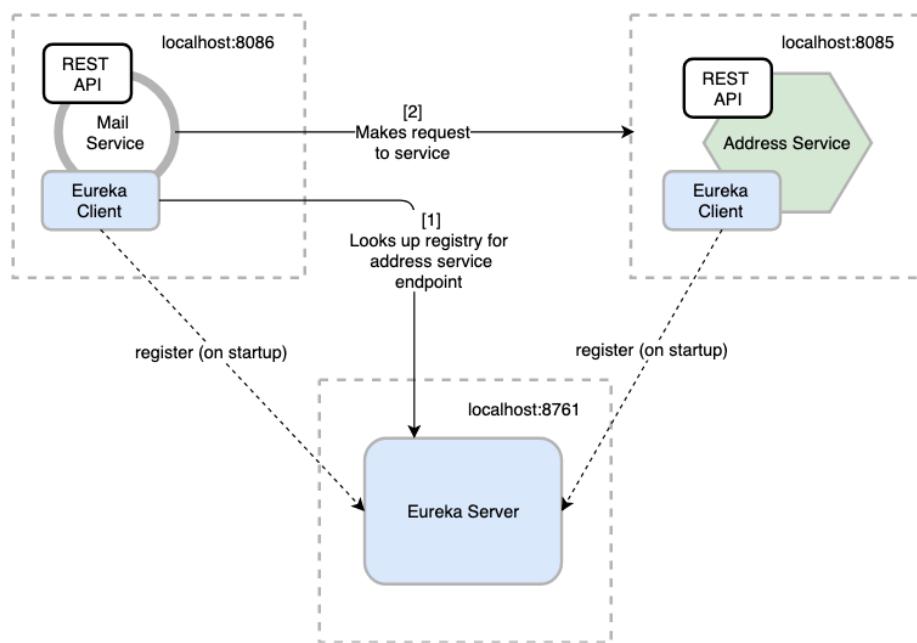


FIGURA 5.5: Flux de descobriment amb Eureka: els microserveis *Mail* i *Address* es registren al servidor; quan el primer necessita localitzar el segon, consulta el registre i obté l'URL actual abans de fer la crida.

La figura 5.5 mostra aquest procés en acció. Quan el *Mail Service* s'inicialitza (quadrad de l'esquerra) envia la seva sol·licitud de registre al servidor Eureka, igual que fa l'*Address Service* (quadrad de la dreta). Un cop registrats, tots dos renoven la seva entrada de forma periòdica. Quan l'usuari demana enviar un correu, el *Mail Service*

no necessita conèixer prèviament on s'està executant l'*Address Service*; simplement interroga el registre, rep l'adreça actual i executa la petició HTTP. Si l'instància canvia de port o se'n desplega una altra, la propera consulta ja retornarà la nova ubicació sense cap intervenció manual. D'aquesta manera, Eureka actua com un directori telefònic dinàmic dels microserveis i assegura que la comunicació interna sigui fluïda, flexible i tolerant a fallades: un pilar imprescindible en la nostra arquitectura distribuïda.

Spring Cloud Gateway

Spring Cloud Gateway és el punt d'entrada reactiu i altament eficient de la nostra arquitectura de microserveis. Actua com a *reverse proxy* basat en Netty: rep totes les peticions externes, avalua predicats i filtres declarats en el `application.yml`, i redirigeix cada sol·licitud al microservei pertinent. Aquesta porta única permet centralitzar l'autenticació amb JWT, aplicar *rate-limiting*, reescriure capçaleres o rutes i, alhora, protegir la xarxa interna de microserveis.

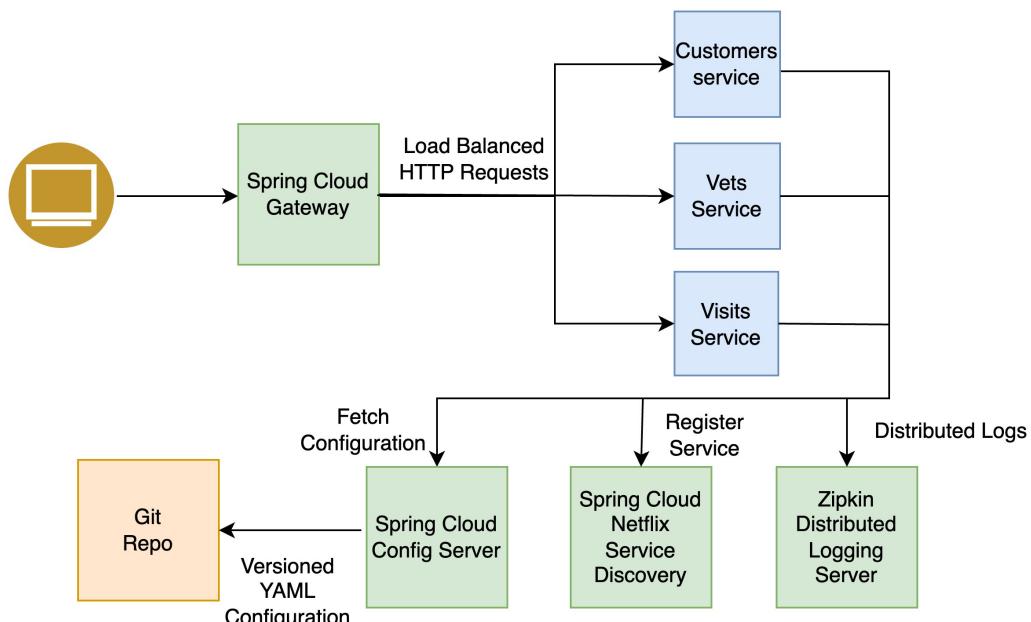


FIGURA 5.6: Flux de Spring Cloud Gateway: totes les peticions externes passen pel *Gateway*, que aplica filtres i distribueix el trànsit cap als microserveis; alhora, es coordina amb el *Config Server* per obtenir la configuració, amb *Service Discovery* per registrar i descobrir instàncies, i amb Zipkin per a la traça distribuïda.

Com il·lustra la figura 5.6, les sol·licituds de l'usuari arriben primer al bloc verd de *Spring Cloud Gateway*, on s'executen predicats (paths, mètodes, host, etc.) i filtres (autenticació, límit de peticions, logging). Si la petició compleix els criteris, el *Gateway* la deriva —amb equilibri de càrrega— al microservei corresponent, que prèviament s'ha registrat al servidor de descoberta. Mentrestant, la configuració dinàmica es recupera del *Config Server* i la traça de cada salt es diposita a Zipkin. D'aquesta manera, aconseguim un punt de control únic, reaccionant amb baixa latència fins i tot sota alta concorrència, i simplificant la seguretat i l'observabilitat de tot l'ecossistema.

PostgreSQL

PostgreSQL és un sistema de gestió de bases de dades relacional d'alt rendiment, de codi obert i reconegut per la seva robustesa, extensibilitat i compliment estricte dels principis ACID (Atomicitat, Consistència, Aïllament i Durabilitat). Ofereix suport avançat per a tipus de dades complexos, incloent-hi columnes JSONB, que és un tipus de dada natiu de PostgreSQL que permet emmagatzemar documents JSON de manera binària i optimitzada, per a l'emmagatzematge eficient de documents semi-estructurats, així com arrays, enums i tipus definits per l'usuari. Permet la creació de consultes SQL complexes, vistes, funcions i triggers, i implementa mecanismes de control de concurrència multiversió (MVCC) per garantir l'accés simultani sense bloquejos. PostgreSQL destaca també per la seva capacitat d'escalar tant verticalment com horitzontalment, la integració amb extensions (com PostGIS per a dades geoespaciales) i el suport natiu per a transaccions distribuïdes. En aquest projecte, PostgreSQL s'executa dins d'un contingidor Docker amb volum persistent, assegurant la integritat i la disponibilitat de les dades fins i tot en cas de reinici o migració de l'entorn.

Spring AMQP i RabbitMQ

RabbitMQ

RabbitMQ és un sistema de missatgeria basat en el protocol AMQP 0-9-1 que introduceix un intermediari (*message broker*) entre productors i consumidors. Aquesta capa intermèdia permet que els microserveis es comuniquin de manera asíncrona i totalment desacoblada: cada missatge s'envia a un *exchange*, que el distribueix a una o més cues (*queues*) segons els *bindings* i les claus de *routing* definides. Gràcies a la persistència de missatges, les confirmacions i els reintents, el broker pot retenir els esdeveniments fins que el consumidor estigui disponible, absorbint pics de càrrega sense perdre informació i permetent escalar productors i consumidors de forma independent.

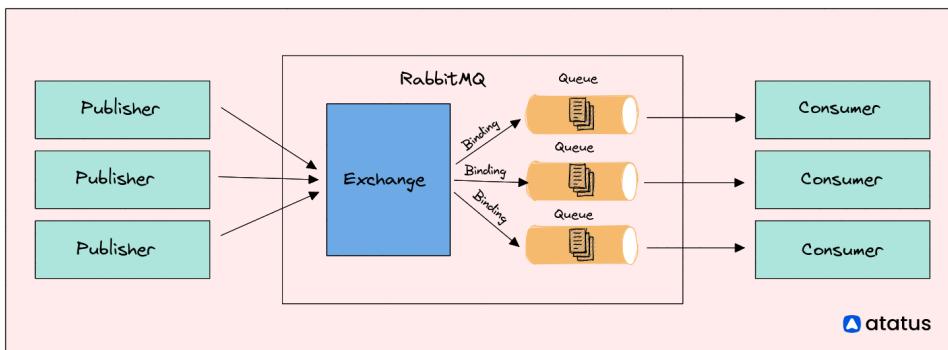


FIGURA 5.7: Esquema de RabbitMQ: un productor publica missatges a l'*exchange*, que els encamina cap a les cues vinculades mitjançant *bindings* i claus de *routing*; cada consumidor processa els missatges de la seva cua de manera independent.

Tal com es veu a la figura 5.7, el productor només necessita conèixer l'*exchange* al qual publicar; un cop el missatge arriba al broker, el ruteig cap a les cues es fa de forma declarativa i dinàmica. Els consumidors, al seu torn, extreuen missatges de les cues que els pertoquen i confirmen la recepció quan els han processat. Aquest patró

elimina dependències rígides entre serveis, facilita l'equilibri de càrrega i garanteix la fiabilitat fins i tot si algun component deixa de respondre momentàniament.

Spring AMQP

Spring AMQP és un conjunt de llibreries de l'ecosistema Spring que proporciona una abstracció d'alt nivell per treballar amb sistemes de missatgeria compatibles amb AMQP, com RabbitMQ. Simplifica la integració amb el broker mitjançant la configuració declarativa de cues, intercanvis i lligams, i ofereix una API orientada a missatges per enviar i rebre dades de manera tipada i segura. Spring AMQP gestiona automàticament la serialització i deserialització d'objectes Java, la gestió de connexions, la confirmació de missatges i la implementació de mecanismes de reintent en cas d'errors temporals. A més, permet definir *listeners* asíncrons que processen missatges de manera concurrent, millorant l'escalabilitat i la reactivitat del sistema. En aquest projecte, la combinació de RabbitMQ i Spring AMQP permet implementar fluxos d'esdeveniments robustos, desacoblar microserveis i garantir la fiabilitat en la comunicació interna.

Spring Cloud OpenFeign

Spring Cloud OpenFeign és una llibreria que permet crear clients REST declaratius a partir d'interfícies Java, simplificant la comunicació entre microserveis. Amb OpenFeign, només cal definir una interfície amb anotacions que descriuen els endpoints remots (@GetMapping, @PostMapping, etc.), i el framework genera automàticament la implementació que realitza les crides HTTP corresponents. Això elimina la necessitat de gestionar manualment la serialització, la construcció d'URLs o la gestió d'errors bàsics, i permet una integració més neta i tipada amb altres serveis.

OpenFeign suporta la injecció d'interceptors personalitzats, que permeten afegir capçaleres comunes (com Authorization amb JWT, X-Request-Id, etc.) a totes les peticions de manera centralitzada. També facilita la gestió de la signatura i validació de tokens JWT, la propagació de contextos de seguretat i la implementació de patrons de resiliència com *retry* o *circuit breaker* mitjançant integració amb Resilience4j. A més, permet configurar codificadors i decodificadors personalitzats per adaptar-se a formats de dades específics (JSON, XML, etc.), i ofereix suport per a la documentació automàtica dels clients.

En aquest projecte, OpenFeign s'utilitza per connectar microserveis interns de manera segura i eficient, garantint la coherència en la gestió de capçaleres, l'autenticació i la traçabilitat de les peticions. Aquesta aproximació declarativa redueix el codi repetitiu, millora la mantenibilitat i facilita l'escalabilitat de l'arquitectura.

Spring Security

Spring Security és el marc de seguretat de referència per a aplicacions Spring. Proporciona mecanismes robustos per a l'autenticació i l'autorització, permetent definir regles d'accés a nivell de rutes, mètodes o recursos. Al nostre projecte, Spring Security s'encarrega de validar cada petició que arriba al Gateway, assegurant que només els usuaris autenticats i amb els permisos adequats puguin accedir als microserveis interns. La configuració es realitza de manera declarativa, integrant filtres personalitzats per a la validació de tokens i la gestió de rols d'usuari.

JWT (JSON Web Token)

JWT és un estàndard per a la transmissió segura d'informació entre parts com a objectes JSON signats digitalment. En aquest sistema, cada usuari rep un token JWT després d'autenticar-se correctament. Aquest token, signat amb un algorisme de criptografia asimètrica (RSA), conté la informació essencial de l'usuari (identitat, rols, data d'expiració, etc.) i s'envia en cada petició posterior al Gateway. L'ús de JWT elimina la necessitat de mantenir estat de sessió al servidor, ja que tota la informació rellevant viatja dins del token, i permet escalar el sistema de manera eficient. A més, la signatura criptogràfica garanteix la integritat i autenticitat del token, reduint la latència i millorant la seguretat global de la plataforma.

5.6 Client d'escriptori

Tauri

Tauri és un framework modern per al desenvolupament d'aplicacions d'escriptori multiplataforma que aprofita tecnologies web (HTML, CSS, JavaScript/TypeScript) per a la interfície d'usuari, però genera binaris natius extremadament lleugers (<15 MB, en comparació amb els >100 MB d'Electron). Utilitza la WebView nativa del sistema operatiu (com WebView2 a Windows, WKWebView a macOS i WebKitGTK a Linux), la qual cosa redueix significativament el consum de memòria i recursos, i minimitza el temps de descàrrega i instal·lació. A més, Tauri proporciona una API segura per accedir a funcionalitats del sistema (fitxers, xarxa, notificacions, etc.) mitjançant un pont entre el codi web i el backend natiu escrit en Rust. Aquesta arquitectura permet mantenir una superfície d'atac reduïda, actualitzacions ràpides i una millor integració amb el sistema operatiu, tot garantint la seguretat i la privacitat de l'usuari.

Svelte

Svelte és un framework per al desenvolupament d'interfícies d'usuari que es distingeix d'altres solucions com React o Vue perquè trasllada la major part del processament al moment de la compilació. En comptes de fer servir un *virtual DOM* i gestionar les actualitzacions de manera reactiva durant l'execució, Svelte transforma els components en codi JavaScript imperatiu optimitzat que manipula directament el DOM. Aquesta aproximació permet que les aplicacions s'iniciin més ràpidament i consumeixin menys recursos, aspectes especialment rellevants en aplicacions d'escriptori on la rapidesa i l'eficiència són fonamentals. Svelte facilita la creació de components reutilitzables, la gestió d'estat reactiu i la integració amb llibreries externes, tot mantenint una sintaxi senzilla i declarativa. En aquest projecte, Svelte s'ha utilitzat per construir la interfície del client d'escriptori, aprofitant la seva eficiència i la facilitat d'integració amb Tauri.

Rust

Rust és un llenguatge de programació de sistemes reconegut per la seva seguretat de memòria, absència de condicions de carrera i alt rendiment. El seu sistema de propietat i préstec (*ownership & borrowing*) evita errors comuns com accessos a memòria nul·la o dobles alliberaments sense necessitat de recollida d'escombraries (*garbage collector*). En el context de Tauri, Rust s'utilitza per implementar el backend natiu

de l'aplicació d'escriptori, gestionant operacions sensibles com l'accés a fitxers, la comunicació amb la xarxa, la criptografia i la integració amb APIs del sistema operatiu. Aquesta elecció garanteix que les operacions crítiques siguin ràpides, segures i fiables, i permet aprofitar l'ecosistema de llibreries de Rust per a funcionalitats avançades. A més, la interoperabilitat entre Rust i el frontend web de Tauri es realitza mitjançant canals segurs, assegurant la separació de privilegis i la protecció davant vulnerabilitats.

5.7 Utilitats addicionals

Docker

Docker és una plataforma de virtualització lleugera que empaqueta cada microser-
vei amb totes les seves dependències dins d'un contenidor aïllat. En comptes de vir-
tualitzar maquinari complet —com fan les màquines virtuals (VM)— Docker aprofi-
ta funcionalitats del nucli de Linux (*namespaces* i *cgroups*) per crear espais d'usuari
independents que comparteixen el mateix nucli del sistema host. D'aquesta manera,
cada contenidor arrenca en pocs segons, ocupa molt menys disc i memòria, i s'execu-
ta de forma idèntica en qualsevol entorn, eliminant el típic "works on my machine".
El procés de construcció, definit al Dockerfile, garanteix traçabilitat i reproduïbilitat:
la mateixa imatge es pot desplegar localment o en producció sense sorpreses.

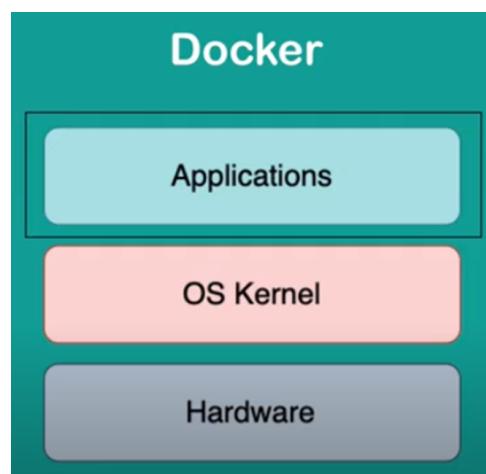


FIGURA 5.8: Arquitectura basada en el nucli compartit: Docker Engine s'executa directament sobre el nucli de Linux i crea contenidors que comparteixen aquest nucli però disposen d'espais de noms i control de recursos propis; per contra, cada VM carrega un sistema ope-
ratiu complet sobre l'hipervisor.

La figura 5.8 mostra com el motor de Docker actua com una capa fina entre el sis-
tema operatiu host i els contenidors. El nucli (en blau) roman únic per a totes les
instàncies, mentre que cada contenidor (en taronja) inclou només la capa d'aplicaci-
ons. Aquesta construcció aprofita que totes les distribucions Linux comparteixen el
mateix nucli, convertint Docker en una "pseudo màquina virtual ultralleugera que
pot crear-se i destruir-se ràpidament, fet que facilita tant l'escalat automàtic com els
desplegaments sense temps mort. En resum, amb Docker garantim portabilitat, con-
sistència i una gestió d'infraestructura tan declarativa com el codi que hi corre. Això

també dóna l'avantatge que es pot crear una configuració única perquè les aplicacions mai tinguin el problema d'estar en un entorn sense alguna configuració o eina que necessitin per funcionar, el que dóna la seguretat que sempre podran funcionar independentment de l'entorn on s'executin.

Docker Compose

Docker Compose és una eina que permet definir i gestionar aplicacions multi-contenidor mitjançant un únic fitxer de configuració YAML (compose.yml). En aquest fitxer es descriuen tots els serveis que formen part de l'arquitectura (per exemple, base de dades, backend, frontend, serveis de missatgeria, etc.), així com les xarxes virtuals i els volums persistents que utilitzen. Compose permet especificar variables d'entorn, dependències entre serveis, ordres d'inicialització i polítiques de reinici, facilitant la coordinació i l'orquestració de tot el sistema. Amb una sola comanda (docker compose up), es poden aixecar tots els contenidors de manera coherent, garantint que la infraestructura es desplega de forma consistent tant en entorns de desenvolupament com de producció. Això simplifica enormement la posada en marxa, les proves i el manteniment de l'ecosistema de microserveis.

5.8 Versionat de dependències

TAULA 5.1: Resum de versions efectives

Tecnologia	Versió
Java	17
Spring Boot	3.2.5
Spring Cloud Gateway	2023.0.1
Spring Security	6.2.4 ¹
PostgreSQL	14.18
RabbitMQ	4.1.2
React	18.2.0
TypeScript	5.2.2
Vite	5.0.0
Tailwind CSS	3.3.5
React Query	5.52.2
Zustand	5.0.4
Tauri	1.5
Svelte	5.0.0
Docker	27.0.3
Docker Compose	3.8

Capítol 6

Requisits del sistema

Aquest capítol descriu de forma detallada tots els requisits que ha de complir el sistema per garantir el seu correcte funcionament. Es classifiquen en requisits funcionals i no funcionals, i es descriuen els aspectes relacionats tant amb el programari com amb el maquinari necessari per al desenvolupament i l'execució.

6.1 Introducció

La plataforma neix amb la voluntat d'ofrir una alternativa lliure, autoallotjada i extensible per a la gestió d'arxius al núvol. Prenent com a punt de partida les motivacions descrites al *Capítol 1*, s'emfatitza la necessitat de controlar completament l'emmagatzematge i la sincronització de dades sense dependre de proveïdors externs.

En síntesi, el sistema comprèn:

- Administració d'arxius i carpetes amb paperera de reciclatge.
- Compartició segura entre usuaris amb permisos granulars.
- Sincronització en temps real entre dispositius mitjançant WebSocket i notificacions d'esdeveniments.

6.2 Requisits funcionals

6.2.1 1. Gestió d'usuaris

- Registre de nous usuaris amb validació de correu electrònic.
- Inici de sessió segur mitjançant autenticació amb JWT.
- Edició i actualització de perfils d'usuari.
- Control d'accés basat en rols: usuari estàndard i administrador.

6.2.2 2. Gestió d'arxius

- Pujar arxius i carpetes a l'espai d'usuari.

- Descarregar arxius de forma individual o en grup.
- Eliminar arxius (amb trasllat a la paperera).
- Crear i gestionar carpetes personals.
- Navegar per l'estructura de directoris.

6.2.3 3. Sistema de paperera

- Eliminació temporal d'arxius amb trasllat a la paperera.
- Restauració d'arxius des de la paperera.
- Eliminació permanent manual o automàtica.

6.2.4 4. Compartició d'arxius

- Definició de permisos d'accés: lectura, escriptura, etc.
- Visualització d'arxius compartits amb altres usuaris.
- Llistat i revocació de comparticions actives.

6.2.5 5. Sincronització en temps real

- Actualització automàtica de canvis entre clients.
- Notificacions d'activitats (pujades, canvis, eliminacions).
- Sincronització d'arxius entre diversos dispositius.

6.2.6 6. Panell d'administració

- Gestió d'usuaris: creació, edició i eliminació.
- Monitoratge del sistema i estat dels serveis.

6.3 Requisits no funcionals

6.3.1 1. Rendiment

- Temps de resposta inferior a 2 segons en operacions habituals. (**Prioritat mitjana**)
- Suport per a la manipulació d'arxius grans (>1GB). (**Prioritat baixa**)
- Optimització de la transferència de dades entre client i servidor. (**Prioritat mitjana**)
- Escalabilitat horitzontal per suportar més càrrega. (**Prioritat alta**)

6.3.2 2. Seguretat

- Autenticació segura amb tokens JWT. (**Prioritat alta**)
- Protecció contra atacs comuns (XSS, CSRF, SQLi). (**Prioritat alta**)
- Validació estricta de dades d'entrada i sortida. (**Prioritat mitjana**)
- Control d'accés granular per arxiu i per usuari. (**Prioritat alta**)

6.3.3 3. Usabilitat

- Interfície intuïtiva i adaptada a dispositius de diverses resolucions. (**Prioritat alta**)
- Funcionalitat de *drag and drop* per facilitar el moviment d'arxius. (**Prioritat alta**)
- Missatges d'error clars i informatius. (**Prioritat alta**)
- Ajuda contextual i indicacions visuals per guiar l'usuari. (**Prioritat mitjana**)

6.3.4 4. Mantenibilitat

- Codi modular per facilitar el manteniment. (**Prioritat mitjana**)
- Arquitectura extensible per afegir noves funcionalitats. (**Prioritat alta**)
- Procés de desplegament fàcil i automatitzat. (**Prioritat mitjana**)
- Diagnòstic i traçabilitat d'errors amb eines de logging. (**Prioritat mitjana**)

6.3.5 5. Compatibilitat

- Suport per a tots els navegadors moderns (Chrome, Firefox, Edge). (**Prioritat alta**)
- Funcionament en sistemes Windows i Linux. (**Prioritat alta**)
- Adaptació a múltiples resolucions de pantalla. (**Prioritat alta**)
- Gestió de formats de fitxer comuns (PDF, DOCX, PNG, etc.). (**Prioritat mitjana**)

6.3.6 6. Escalabilitat

- Arquitectura distribuïda basada en microserveis. (**Prioritat alta**)
- Gestió eficient de recursos i instàncies. (**Prioritat mitjana**)
- Optimització de l'espai d'emmagatzematge. (**Prioritat baixa**)
- Capacitat per a múltiples usuaris actius simultàniament. (**Prioritat mitjana**)

6.3.7 7. Portabilitat

- Instal·lació senzilla a través de contenidors Docker. (**Prioritat alta**)
- Configuració mitjançant fitxers .env editables. (**Prioritat alta**)
- Dependències mínimes per executar cada component. (**Prioritat alta**)
- Documentació clara per a desenvolupadors i usuaris. (**Prioritat alta**)
- Guia d'instal·lació i configuració pas a pas. (**Prioritat alta**)

6.4 Requisits de maquinari i programari

Per tal d'executar la solució completa –format microserveis basats en Spring Boot, PostgreSQL i RabbitMQ al costat del frontend React i del client d'escriptori Tauri–, cal comptar amb els recursos següents:

- **Entorn servidor/desenvolupament (Docker Compose)**: un processador de com a mínim quatre nuclis (Intel Core i5 o equivalent), entre **8 GB i 16 GB** de memòria RAM i **5 GB** d'espai lliure per allotjar imatges i volums de dades. Es recomana un sistema operatiu de 64 bits amb Docker 20+ i Docker Compose 2+.
- **Client d'escriptori (Tauri + Svelte)**: gràcies a la compilació a codi natiu, l'aplicació és molt lleugera i funciona sense problemes en equips de 64 bits amb **2 GB** de RAM i uns **200 MB** d'espai en disc. Compatible amb Windows, Linux i macOS.
- **Generació de clients** (web i escriptori): cal disposar de **Node.js 18+, Rust 1.73+** i Docker per orquestrar els serveis durant el desenvolupament.

6.5 Resum

Aquest capítol ha recollit tots els requisits –funcionals i no funcionals– que defineixen el producte. S'han desglossat les funcionalitats clau, els estàndards de seguretat i els objectius de rendiment, així com els requisits concrets de maquinari i programari obtinguts de l'anàlisi del codi dels directoris @server i @ui-new. Els capítols posteriors de disseny i implementació demostren com s'han abordat aquests requisits, tot i que algun objectiu, com la gestió d'arxius compartits al client d'escriptori, es va haver de descartar per la seva complexitat i falta de temps.

Capítol 7

Estudis i decisions

7.1 Introducció

Aquest capítol explora el procés de reflexió i les decisions tècniques que han donat forma a l'arquitectura d'aquest projecte. Més que una simple llista d'eines, el que es presenta a continuació és un recorregut per les anàlisis i els criteris que van guiar l'elecció de cada component, sempre amb l'objectiu de materialitzar la visió central del projecte: crear una plataforma de gestió de fitxers autogestionada, segura i de codi obert, on l'usuari final mantingui el control total sobre les seves dades.

Per navegar aquest procés de decisió, em vaig basar en un conjunt de criteris clau:

- **Experiència prèvia:** Aprofitar un coneixement de base en certes tecnologies per accelerar les fases inicials del desenvolupament.
- **Rendiment i eficiència:** Prioritzar solucions lleugeres, un requisit crític per al client d'escriptori.
- **Cohesió de l'ecosistema:** Optar per conjunts d'eines dissenyades per integrar-se de manera fluida.
- **Capacitats tècniques específiques:** Seleccionar tecnologies que oferissin solucions a reptes concrets com la comunicació asíncrona o la gestió de dades.
- **Codi obert i compliment legal:** Escollir eines de codi obert, un factor que garanteix la sostenibilitat i la viabilitat econòmica del projecte a llarg termini i que facilitessin l'adhesió a normatives com el RGPD.

Aquests directrius em van permetre dissenyar una arquitectura coherent i alineada amb la visió del projecte.

7.2 Pila Tecnològica del Backend

Per al backend, vaig decidir implementar una arquitectura de microserveis. Aquesta elecció es fonamenta en la necessitat de construir un sistema modular, escalable i fàcil de mantenir, on cada servei tingui una responsabilitat única. Vaig escollit l'ecosistema de Java i Spring per la seva coneguda maduresa, estabilitat i ampli suport en entorns empresarials.

7.2.1 Spring Boot

Spring Boot és un framework que simplifica la creació d'aplicacions Java autònomes i llestes per a producció. Facilita la configuració i el desplegament de microserveis gràcies a la seva filosofia de convenció sobre configuració i a la inclusió de servidors web embeguts.

Abans de prendre una decisió, vaig analitzar alternatives modernes a Spring Boot altament considerades en el desenvolupament de microserveis per la seva eficiència [1]. A continuació, es presenta una taula comparativa que resumeix els factors clau.

Framework	Avantatges (Pros)	Inconvenients (Contres)
Spring Boot	<ul style="list-style-type: none"> Ecosistema molt madur i estable. Gran comunitat i suport empresarial. Experiència prèvia que garantia alta productivitat. 	<ul style="list-style-type: none"> Temps d'arrencada més lent. Major consum de memòria en comparació amb les alternatives.
Quarkus	<ul style="list-style-type: none"> Temps d'arrencada extremadament ràpids. Baix consum de memòria (optimitzat per a contenidors). Enfocament natiu a GraalVM. 	<ul style="list-style-type: none"> Corba d'aprenentatge en no tenir experiència prèvia. Risc per al compliment dels terminis del projecte.
Micronaut	<ul style="list-style-type: none"> Temps d'arrencada quasi instantanis. Mínim ús de reflexió gràcies a la injecció de dependències en temps de compilació. 	<ul style="list-style-type: none"> També presentava una corba d'aprenentatge. Ecosistema menys extens que el de Spring.

TAULA 7.1: Comparativa de frameworks Java per a microserveis.

Justificació de l'elecció Vaig basar la meva elecció principalment en l'experiència prèvia que ja tenia amb l'ecosistema Spring. Aquesta familiaritat em va permetre assolir una alta productivitat des de l'inici del projecte. Tot i que vaig considerar alternatives modernes com Quarkus o Micronaut, que prometen millors temps d'arrencada, la corba d'aprenentatge associada hauria suposat un risc per al compliment dels terminis. Spring Boot representava, doncs, la via més pragmàtica i segura per al meu cas.

7.2.2 Spring Data JPA

Spring Data JPA facilita la implementació de la capa de persistència de dades en aplicacions Spring, abstraient gran part del codi necessari per a les operacions de base de dades. La

seva capacitat més destacada és la generació automàtica de consultes a partir de la signatura dels mètodes en una interfície de repositori.

Justificació de l'elecció Vaig adoptar Spring Data JPA per la seva integració nativa i sense fricció amb Spring Boot. L'objectiu era accelerar el desenvolupament de la capa de dades, i la seva capacitat per generar repositoris em permetia eliminar una gran quantitat de codi repetitiu en comparació amb l'ús de JPA estàndard o JDBC. A més, en fomentar l'ús de consultes parametritzades de manera inherent, proporciona una capa de seguretat fonamental en prevenir atacs de tipus injecció SQL, un requisit indispensable per garantir la integritat de les dades segons el RGPD.

7.2.3 PostgreSQL

PostgreSQL és un sistema de gestió de bases de dades relacional d'objectes, reconegut per la seva robustesa, extensibilitat i compliment estricte de l'estàndard SQL i les propietats ACID.

Per a la selecció del sistema de gestió de bases de dades, vaig comparar les opcions més consolidades del mercat [2], tenint en compte tant requisits tècnics com factors pràctics.

Justificació de l'elecció Vaig escollir PostgreSQL no només per la seva reputació com a base de dades fiable i de codi obert, sinó també perquè és el sistema amb el qual tinc més experiència, la qual cosa em va permetre ser més productiu. A més, la seva coneguda robustesa en la gestió de rols i permisos s'alineava perfectament amb les exigències del RGPD per assegurar la confidencialitat i la integritat de les dades dels usuaris, sent aquest un factor clau en la decisió.

7.2.4 Ecosistema Spring Cloud

Una arquitectura de microserveis requereix un conjunt d'eines per gestionar la comunicació, el descobriment i l'enrutament de serveis. Vaig optar per la suite de Spring Cloud per garantir una **màxima cohesió i compatibilitat**, ja que les seves eines estan dissenyades per funcionar conjuntament de manera nativa.

Spring Cloud Gateway

Spring Cloud Gateway actua com la porta d'enllaç (API Gateway) del sistema. La seva funció principal és ser l'únic punt d'entrada per a totes les peticions externes. Aquesta centralització és clau per a la seguretat i l'organització de l'arquitectura.

Justificació i ús en el projecte En aquest projecte, el Gateway s'encarrega de:

- **Enrutament dinàmic:** Dirigir cada petició al microservei corresponent (gestió de fitxers, usuaris, etc.) basant-se en el camí de l'URL.
- **Seguretat centralitzada:** Integrar-se amb Spring Security per aplicar un filtre d'autenticació a cada petició. Abans que una sol·licitud arribi a un servei intern, el Gateway valida el token JWT, garantint que només els usuaris autenticats tinguin accés. Això simplifica la lògica dels microserveis, que no necessiten implementar aquesta validació individualment.

SGBD	Avantatges (Pros)	Inconvenients (Contres)
PostgreSQL	<ul style="list-style-type: none"> • Codi obert i gratuït. • Altament extensible i personalitzable. • Suport avançat per a tipus de dades complexos (JSONB, GIS, etc.). • Compliment estricte de l'estàndard SQL. 	<ul style="list-style-type: none"> • Pot tenir una corba d'aprenentatge lleugerament superior a MySQL per a tases bàsiques.
MySQL	<ul style="list-style-type: none"> • Molt popular i fàcil d'iniciar per a aplicacions senzilles. • Bon rendiment en escenaris de lectura intensiva. • Gran comunitat d'usuaris. 	<ul style="list-style-type: none"> • Menys funcionalitats avançades que PostgreSQL. • Propietat d'Oracle, la qual cosa genera incertesa sobre el seu futur com a projecte totalment obert.
Oracle	<ul style="list-style-type: none"> • Solució empresarial molt potent i amb un ampli ventall de funcionalitats. • Suport tècnic professional garantit pel proveïdor. 	<ul style="list-style-type: none"> • Cost de llicències extremadament elevat. • Propietari, la qual cosa genera una forta dependència (vendor lock-in). • Complexitat elevada en la seva administració.

TAULA 7.2: Comparativa de Sistemes de Gestió de Bases de Dades.

L'elecció del Gateway va ser fonamental per crear una barrera de seguretat robusta i mantenir l'ordre en un sistema distribuït.

Eureka

Eureka és un servidor de descobriment de serveis (Service Discovery). En un entorn de microserveis, les instàncies poden aparèixer i desaparèixer dinàmicament. Eureka soluciona el problema de com un servei pot trobar la ubicació de xarxa (IP i port) d'un altre.

Justificació i ús en el projecte Cada microservei es registra a Eureka en arrencar. Quan un servei necessita comunicar-se amb un altre, consulta a Eureka per obtenir la seva ubicació actualitzada. Això aporta:

- **Resiliència i escalabilitat:** El sistema pot escalar horitzontalment o sobreviure a la caiguda d'una instància sense necessitat de reconfiguració manual.
- **Desacoblament:** Els serveis no necessiten conèixer les adreces físiques dels altres, simplificant la configuració i el desplegament.

Spring Cloud OpenFeign

OpenFeign és un client REST declaratiu que simplifica la comunicació entre serveis. Permet definir la comunicació amb una API REST remota simplement creant una interfície de Java i anotant-la.

Justificació i ús en el projecte En lloc d'escriure manualment el codi per a peticions HTTP, OpenFeign ho automatitza. S'utilitza, per exemple, quan el servei de gestió de fitxers necessita dades del servei d'usuaris. Això ofereix:

- **Codi més net i lleigible:** La lògica de la petició HTTP queda abstracta darrere d'una simple interfície, reduint el codi repetitiu.
- **Integració nativa:** Es connecta automàticament amb Eureka per resoldre els noms dels serveis i realitzar balanceig de càrrega.

L'ús d'OpenFeign va accelerar el desenvolupament i va millorar la mantenibilitat del codi de comunicació interna.

7.2.5 Spring Security

Spring Security és un framework potent i altament personalitzable que proporciona funcionalitats d'autenticació i control d'accés per a aplicacions Java.

Justificació de l'elecció La implementació d'un sistema d'autenticació robust era un requisit no funcional crític, directament lligat al compliment del RGPD. Vaig escollit Spring Security per la seva maduresa i la seva integració completa amb Spring Boot. La seva implementació en el servei UserAuthentication, juntament amb el filtre del gateway, em va permetre configurar un flux d'autenticació segur basat en JWT, amb emmagatzematge de contrasenyes mitjançant l'algorisme BCrypt, garantint que les dades d'accés dels usuaris estiguin protegides en tot moment.

7.2.6 RabbitMQ

RabbitMQ és un intermediari de missatges (message broker) que implementa el protocol AMQP, dissenyat per gestionar la comunicació asíncrona entre diferents components d'un sistema.

Per a la comunicació asíncrona, vaig avaluar dos dels intermediaris de missatges més populars [3].

Justificació de l'elecció L'elecció d'un intermediari de missatges es va centrar en resoldre un repte específic: la comunicació asíncrona per millorar la resiliència. Vaig escollit **RabbitMQ** per la seva **simplicitat en la configuració i gestió**. El seu ús, per exemple, en l'eliminació de dades en cascada, no només millora l'experiència d'usuari, sinó que també garanteix la fiabilitat en el compliment de les sol·licituds

Sistema	Avantatges (Pros)	Inconvenients (Contres)
RabbitMQ	<ul style="list-style-type: none"> Configuració i gestió relativament senzilles. Gran flexibilitat en l'enrutament de missatges. Ideal per a patrons de desacoblament i tasques en segon pla. 	<ul style="list-style-type: none"> Menor rendiment que Kafka en escenaris de volum de dades massiu. L'emmagatzematge no està dissenyat per a una retenció de dades a llarg termini per defecte.
Apache Kafka	<ul style="list-style-type: none"> Rendiment extremadament alt i alta escalabilitat. Sistema de log distribuït i persistent, ideal per a <i>event sourcing</i>. 	<ul style="list-style-type: none"> Configuració, gestió i operació notablement més complexes. Corba d'aprenentatge més pronunciada.

TAULA 7.3: Comparativa d'Intermediaris de Missatges.

d'eliminació de dades sota el RGPD, assegurant que l'operació es completi de manera fiable fins i tot si un servei falla temporalment.

L'objectiu no era aplicar la comunicació asíncrona a tot el sistema, sinó utilitzar-la de manera estratègica. Concretament, es va implementar en processos com l'eliminació de dades en cascada entre microserveis. Aquest enfocament permet que la petició principal de l'usuari (p. ex., eliminar un fitxer) es completi ràpidament sense quedar bloquejada esperant la neteja de dades dependents. A més, proporciona un mecanisme de reintent transparent: si un servei consumidor falla temporalment, el sistema té un mecanisme de reintent asíncron que garanteix la consistència final de les dades sense que l'usuari ho percebi ni hagi d'intervenir. Per aquest cas d'ús, la facilitat d'implementació de RabbitMQ era ideal.

7.3 Pila Tecnològica del Frontend Web

Per al client web, necessitava una solució moderna que permetés construir una interfície d'usuari interactiva i eficient.

7.3.1 React

React és una biblioteca de JavaScript per construir interfícies d'usuari, basada en un model de components reutilitzables i un flux de dades unidireccional. El seu ús del Virtual DOM optimitza les actualitzacions de la interfície i millora el rendiment en aplicacions complexes.

A l'hora de seleccionar la tecnologia per al frontend, vaig comparar les biblioteques i frameworks més rellevants del mercat.

Justificació de l'elecció L'elecció de React, tot i basar-se en una familiaritat prèvia, no va ser una decisió superficial. Més enllà de l'experiència inicial, vaig valorar positivament la seva **flexibilitat** i el seu **enfocament en el rendiment**. L'arquitectura basada en components i l'ús del Virtual DOM s'alineaven amb el meu objectiu de construir una interfície eficient i modular. A diferència d'Angular, que imposa una estructura més rígida, React em proporcionava la llibertat de seleccionar les millors eines per a cada necessitat específica (com React Query per a l'estat del servidor i Zustand per a l'estat global). A més, el seu vast ecosistema de llibreries i el gran suport de la comunitat em donava la confiança necessària per afrontar els reptes del projecte, sabent que disposaria de solucions provades per la indústria.

7.3.2 React Query (TanStack Query)

React Query és una llibreria per a la gestió de l'estat del servidor (server state) en aplicacions React. S'encarrega de la consulta, la gestió de memòria cache i la sincronització de dades amb fonts externes com una API.

Justificació de l'elecció Per a la comunicació amb el backend, vaig voler evitar la gestió manual de l'estat de les dades amb *hooks* com *useState* i *useEffect*, ja que pot portar a codi complex i propens a errors. Vaig triar React Query amb la intenció de disposar d'una solució més robusta i declarativa. El meu objectiu en adoptar-la era simplificar la lògica de peticions asíncrones, millorar l'experiència d'usuari amb estratègies de memòria cache intel·ligents i, en definitiva, mantenir la interfície sincronitzada amb el backend de manera eficient.

7.3.3 Zustand

Zustand és una llibreria minimalistica per a la gestió d'estat global (client state) en React, basada en una API senzilla de hooks.

Justificació de l'elecció Per l'estat global de la interfície (elements no dependents del servidor, com la selecció d'arxius), vaig avaluar diverses opcions. Vaig descartar Redux per considerar-lo excessivament complex per a les meves necessitats. Vaig escollit Zustand perquè buscava una solució lleugera, ràpida i amb una corba d'aprenentatge mínima. La seva simplicitat i el seu enfocament basat en *hooks* em semblaven ideals per gestionar l'estat global necessari sense afegir una sobrecàrrega de configuració ni afectar negativament el rendiment de l'aplicació.

7.4 Pila Tecnològica del Client d'Escriptori

Per al client d'escriptori, el requisit principal era aconseguir una aplicació multiplataforma que fos nativa en rendiment i consum de recursos.

7.4.1 Tauri i Svelte

Tauri és un framework que permet construir aplicacions d'escriptori utilitzant tecnologies web per a la interfície i un backend natiu escrit en Rust. Svelte, per la seva banda, és un compilador que transforma components d'interfície en codi JavaScript imperatiu altament optimitzat.

Justificació de l'elecció La meva decisió clau en aquest àmbit va ser **prioritzar el rendiment i la lleugeresa**. Vaig descartar l'alternativa més estesa, Electron, a causa del seu conegut alt consum de memòria i la gran mida dels binaris que genera. Vaig escollit **Tauri** perquè la seva arquitectura, basada en la *WebView* nativa del sistema operatiu, em permetia crear una aplicació molt més eficient i amb una mida final significativament menor.

De manera complementària, per a la interfície d'aquesta aplicació d'escriptori, vaig optar per **Svelte** en lloc de React. Atès que Svelte és un compilador, genera un codi final molt més petit i eficient, sense la sobrecàrrega d'un DOM virtual en temps d'execució. Aquesta combinació de Tauri i Svelte em sembla la idònia per assolir el meu objectiu d'una aplicació d'escriptori ràpida, lleugera i amb una experiència d'usuari fluida.

La decisió per aixo va tenir un cost que no preveia en la corva d'aprenentage de rust, que va ser mes gran del que preveia. Si be encara ara crec que va ser una bona decisió, no puc assegurar que en cas de tornar a començar el projecte el tornaria a triar.

7.5 Maquinari i infraestructura

Durant el desenvolupament del projecte s'ha utilitzat un equip amb les especificacions següents:

- **CPU:** Intel Core i7-1185G7 (11a generació) amb 4 nuclis (8 fils) a 3.00 GHz.
- **Memòria RAM:** 32 GB.
- **Sistema operatiu:** Ubuntu 22.04.4 LTS (Linux).

Pel desplegament es recomana executar-la en qualsevol servidor o màquina virtual amb mínim 2 nuclis de CPU, 4 GB de RAM i 50 GB d'emmagatzematge, amb accés a internet i un *reverse proxy* per gestionar els certificats SSL. El reverse proxy no és una obligatorietat tècnica perquè el projecte funcioni, però és la solució estàndard i recomanada per garantir la seguretat. No s'ha afegit al projecte la configuració d'un reverse proxy perquè el temps disponible no permetia abordar la recerca necessària per desplegar-ne un de manera segura. Es planteja com una millora futura, ja que per complir amb la normativa de protecció de dades (RGPD), que exigeix garantir la confidencialitat de les dades personals en trànsit, és imprescindible implementar el xifratge SSL/TLS. L'ús d'un reverse proxy és la via més eficient i robusta per assolir aquest objectiu.

Les versions específiques recomanades per als serveis de tercers són **PostgreSQL 16 o superior** i **RabbitMQ 3.12 o superior**, ja que són les versions estables més recents amb les quals s'ha provat el sistema durant el desenvolupament. L'ús de versions anteriors podria causar incompatibilitats o comportaments inesperats en alguns microserveis.

Els requisits detallats estan al capítol 6.

7.6 Eines de disseny i estil (UI/UX)

Per al disseny es va optar per Tailwind CSS per agilitzar el desenvolupament. Les llibreries complementàries utilitzades són:

- **Radix UI:** Biblioteca de components d'interfície sense estils, de baix nivell i accessibles, que serveix com a base per construir un sistema de disseny personalitzat.
- **Remix Icon i React Icons:** Col·leccions d'ícones SVG de codi obert, fàcilment integrables en projectes React per millorar la usabilitat visual.
- **Tailwind Merge, clsx, Class Variance Authority:** Utilitats per gestionar i fusionar classes de Tailwind CSS de manera intel·ligent, evitant conflictes d'estils i simplificant la lògica de variants en components.
- **Dnd-kit/core:** Conjunt d'eines lleuger i modular per crear funcionalitats d'arróssegar i deixar anar ('drag-and-drop') accessibles i performants a React.
- **Selecto.js:** Llibreria per seleccionar elements mitjançant el ratolí o el tacte, utilitzada per implementar la selecció múltiple d'arxius i carpetes a la interfície web.

7.7 Cadena d'eines i DevOps

S'han utilitzat les eines següents per a la construcció i el desplegament:

- **Backend:** Maven 3.9.x (Maven Wrapper).
- **Frontend:** Node.js v20.14.0 amb pnpm.
- **Contenidors:** Docker Engine 27.0.3 i Docker Compose v2.28.1.

Per a la orquestració dels contenidors del projecte, s'utilitza un fitxer docker-compose.yml. Es va optar per aquesta solució per la seva simplicitat, ja que satisfà les necessitats del sistema amb una configuració senzilla que es pot executar en qualsevol ordinador amb un script, basant-se en el coneixement ja adquirit sobre la tecnologia.

Com a treball a futur, s'ha previst afegir fitxers de configuració de Helm que permetrien instal·lar el sistema en una infraestructura Kubernetes per a usuaris més avançats que ho puguin requerir, oferint més escalabilitat i robustesa.

El projecte és a GitHub, la qual cosa facilita aquesta futura integració. A més, en ser un repositori públic, facilita la col·laboració amb altres desenvolupadors i, tal com mostren les estadístiques, és l'eina més popular per a la gestió de projectes de codi obert [4].

7.8 Llicències i compliment legal

La llicència escollida és MIT per màxima flexibilitat:

7.9 Traçabilitat global amb els requisits

La Taula 7.6 resumeix la traçabilitat entre els criteris de decisió, la tecnologia escollida i els requisits que cobreix cadascuna.

Framework/Biblioteca	Avantatges (Pros)	Inconvenients (Contres)
React	<ul style="list-style-type: none"> • Arquitectura basada en components que fomenta la reutilització. • Alt rendiment gràcies al Virtual DOM. • Ecosistema de llibreries enorme i gran suport de la comunitat. • Flexibilitat per triar les eines complementàries (p. ex., gestió d'estat, enrutament). 	<ul style="list-style-type: none"> • És una biblioteca, no un framework complet, la qual cosa requereix integrar altres solucions. • La llibertat d'elecció pot portar a una major complexitat en la configuració inicial.
Angular	<ul style="list-style-type: none"> • Framework complet i robust amb solucions integrades ("out-of-the-box"). • Basat en TypeScript, que millora la mantenibilitat del codi. • Fort suport empresarial per part de Google. 	<ul style="list-style-type: none"> • Corba d'aprenentatge més pronunciada i major verbositat. • Menys flexible a causa de la seva naturalesa més rígida i opinionada.
Vue	<ul style="list-style-type: none"> • Corba d'aprenentatge molt suau i excel·lent documentació. • Bon rendiment i flexibilitat. 	<ul style="list-style-type: none"> • Comunitat i ecosistema més petits en comparació amb React. • Menys presència en grans aplicacions empresarials.

TAULA 7.4: Comparativa de tecnologies per al Frontend Web.

Llicència	Ús comercial	Publicar derivats	Compatibilitat propietària
MIT (escollida)	Sí	No	Molt alta
Apache-2.0	Sí	Parcial	Alta
GPL-3.0	Sí	Obligatori	Baixa
LGPL-3.0	Sí	Parcial	Mitjana

TAULA 7.5: Comparativa abreujada de llicències

Criteri Inicial	Tecnologia Escollida	Requisit cobert
Experiència prèvia	Spring Boot, Maven, React	RF-1 a RF-10, RNF-Rendiment, RNF-Mantenibilitat
Rendiment i eficiència	Tauri, Svelte, Tailwind	RNF-Rendiment, RNF-Compatibilitat, RNF-Usabilitat
Cohesió ecosistema	Spring Cloud, Docker Compose	RNF-Escalabilitat, RNF-Portabilitat
Capacitats tècniques	RabbitMQ, PostgreSQL	RF-Gestió dades, RNF-Escalabilitat
Codi obert	GitHub, MIT License	Viabilitat econòmica, Legalitat

TAULA 7.6: Traçabilitat global criteris-tecnologia-requisits

Capítol 8

Anàlisi i disseny del sistema

8.1 Introducció

Aquest capítol presenta l'anàlisi funcional i estructural del sistema desenvolupat, així com les decisions arquitectòniques preses durant la seva fase de disseny. Es descriuen els components principals de l'arquitectura de microserveis, els fluxos de dades, els casos d'ús més rellevants i els esbossos inicials de la interfície d'usuari, tant pel client web com per al client d'escriptori.

8.2 Anàlisi funcional

8.2.1 Actors i mòduls

Els principals actors identificats en el sistema, ordenats per jerarquia, són:

- **Client Tauri:** Representa l'usuari de l'aplicació d'escriptori. Té accés a totes les funcionalitats de gestió de fitxers i la capacitat de sincronització automàtica amb una carpeta local.
- **Usuari web:** Usuari que interactua amb el sistema a través de la interfície web. Pot gestionar els seus arxius, compartir-los i configurar el seu compte.
- **Administrador:** A més de les capacitats d'un usuari normal, pot gestionar altres usuaris, incloent la modificació de dades i l'eliminació de comptes.
- **Superadministrador:** El rol amb màxims privilegis. Té control total sobre la plataforma, incloent la gestió de comptes d'administradors i la capacitat de restablir qualsevol contrasenya.

El sistema es divideix funcionalment en els següents mòduls:

- Autenticació i gestió d'usuaris
- Gestió d'arxius i carpetes
- Control d'accés
- Compartició d'arxius
- Sincronització en temps real

- Gestió de la paperera

8.2.2 Casos d'ús principals

A continuació es descriuen diversos casos d'ús extrets de les fitxes funcionals, detaillant la interacció entre els principals serveis implicats:

UC-01: Registrar-se

- **Descripció:** L'usuari crea un compte nou.
- **Actors:** Usuari.
- **Precondicions:** No haver iniciat sessió.
- **Postcondicions:** Compte creat i sessió iniciada.
- **Escenari principal:**
 1. El client envia una sol·licitud de registre al servei **UserAuthentication** a través del Gateway.
 2. El servei valida les dades rebudes:
 - Comprova que el format del nom d'usuari, contrasenya i email siguin correctes.
 - Verifica que el nom d'usuari no estigui ja en ús.
 - Consulta a **UserManagement** per assegurar que l'email no estigui registrat.
 3. Si les validacions són correctes, **UserAuthentication** guarda les credencials (nom d'usuari i contrasenya xifrada) i inicia la creació de dades en altres serveis:
 - Fa una crida a **UserManagement** per guardar la informació personal de l'usuari (email, nom i cognoms).
 - Crida a **FileManagement** per crear la carpeta arrel de l'usuari.
 - Finalment, envia un missatge asíncron a través de RabbitMQ a **Sync-Service** per generar l'estat inicial de sincronització (*snapshot*) amb la carpeta arrel.
 4. Un cop finalitzat el procés, **UserAuthentication** retorna els tokens d'accés i de refresh per iniciar la sessió automàticament.

UC-02: Iniciar sessió

- **Descripció:** L'usuari inicia sessió amb el seu nom i contrasenya.
- **Actors:** Usuari.

- **Precondicions:** Tenir un compte vàlid.
- **Postcondicions:** Sessió activa.
- **Escenari principal:**
 1. El client envia el nom d'usuari i la contrasenya al servei **UserAuthentication**.
 2. El servei busca l'usuari a la base de dades a partir del seu nom.
 3. Si l'usuari existeix, compara la contrasenya rebuda amb la versió xifrada emmagatzemada.
 4. Si les credencials són correctes, genera un nou token d'accés (JWT) de curta durada i un token de refresh de llarga durada.
 5. Finalment, retorna els dos tokens al client per iniciar la sessió i mantenir-la activa.

UC-08: Pujar un arxiu

- **Descripció:** Pujada de fitxers o creació de carpetes.
- **Actors:** Usuari.
- **Precondicions:** Permís d'escriptura a la carpeta de destinació.
- **Postcondicions:** Nou element emmagatzemat i notificat.
- **Escenari principal:**
 1. El Gateway valida el token JWT i reenvia la petició a **FileManagement**.
 2. El servei consulta a **FileAccessControl** per verificar que l'usuari té permís d'escriptura (WRITE) a la carpeta de destinació.
 3. Si el permís és correcte, crea les metadades de l'arxiu (nom, mida, etc.) a la seva base de dades.
 4. Emmagatzema el contingut del fitxer al sistema d'arxius del servidor, utilitzant un ID únic com a nom.
 5. Sol·licita a **FileAccessControl** que assigni el permís de propietari (ADMIN) sobre el nou element a l'usuari que l'ha pujat.
 6. Finalment, envia un missatge asíncron a **SyncService** per notificar la creació i actualitzar els clients.

UC-10: Descarregar un arxiu

- **Descripció:** Obtenir el contingut d'un arxiu o carpeta.
- **Actors:** Usuari.

- **Precondicions:** Permis de lectura sobre l'element sol·licitat.
- **Postcondicions:** Arxiu descarregat pel client.
- **Escenari principal:**
 1. El Gateway valida el JWT i reenvia la petició a **FileManagement**.
 2. Aquest consulta **FileAccessControl** per confirmar que l'usuari té permís de lectura (READ).
 3. Si els permisos són correctes, recupera el fitxer del sistema d'emmagatzematge. Si és una carpeta, la comprimeix en format ZIP.
 4. Retorna el contingut com un flux de dades (*stream*) perquè el client iniciï la descàrrega.

UC-11: Moure un arxiu a la paperera

- **Descripció:** Moure elements a la paperera.
- **Actors:** Usuari.
- **Precondicions:** Permis d'escriptura sobre l'element.
- **Postcondicions:** Element marcat com a eliminat i visible a la paperera.
- **Escenari principal:**
 1. El Gateway valida el JWT i envia la petició a **TrashService**.
 2. **TrashService** verifica a **FileAccessControl** que l'usuari té permís d'escriptura.
 3. Crida a **FileManagement** perquè marqui l'element i els seus descendents com a eliminats (sense esborrar-los físicament).
 4. Crea un registre a la seva pròpia base de dades (TrashRecord) per cada element mogut, emmagatzemant la data d'eliminació i de caducitat.
 5. **FileManagement** notifica a **SyncService** el canvi d'estat per actualitzar els clients.

UC-12: Eliminar definitivament

- **Descripció:** Esborrar definitivament un element de la paperera.
- **Actors:** Usuari.
- **Precondicions:** Element a la paperera i ser-ne el propietari.
- **Postcondicions:** Element eliminat de forma permanent.
- **Escenari principal:**

1. El Gateway envia la petició a **TrashService**.
2. El servei verifica que l'usuari és el propietari de l'element.
3. Crida a **FileManagement** per esborrar l'arxiu físic i les seves metadades.
4. Canvia l'estat del TrashRecord a PENDING_DELETION i inicia un procés de purga asíncron.
5. A través de missatges per cua, ordena a **FileAccessControl** i **FileSharing** que eliminin totes les regles associades a l'element.
6. Un cop confirmat per tots els serveis, el TrashRecord s'esborra.
7. Es notifica a **SyncService** per eliminar les còpies locals de l'element.

UC-13: Compartir arxiu

- **Descripció:** Concedir o revocar accessos sobre elements a altres usuaris.
- **Actors:** Usuari.
- **Precondicions:** Permís de propietari o d'administrador sobre l'element.
- **Postcondicions:** Els usuaris seleccionats obtenen o perden l'accés indicat.
- **Escenari principal:**
 1. El Gateway valida el JWT i passa la petició a **FileSharing**.
 2. El servei verifica a **FileAccessControl** que el sol·licitant és el propietari (ADMIN) de l'element.
 3. Consulta a **UserManagement** per obtenir l'ID de l'usuari amb qui es vol compartir.
 4. Sol·licita a **FileAccessControl** que creï una nova regla d'accés (lectura o escriptura) per a l'usuari convidat sobre l'element i els seus descendents (si es una carpeta).
 5. Desa un registre de la compartició a la seva base de dades.
 6. Notifica a **SyncService** a través de RabbitMQ per propagar els canvis als clients implicats.

(La resta de fitxes completes es poden consultar a l'Apèndix A)

8.2.3 Matriu de traçabilitat entre requisits i casos d'ús

Per garantir que tots els requisits funcionals descrits al Capítol 6 estan coberts per la funcionalitat del sistema, es presenta a la Taula 8.1 una matriu de traçabilitat detallada. Aquesta relaciona cada requisit funcional amb els casos d'ús individuals (detallats a l'Apèndix A) que l'implementen.

TAULA 8.1: Matriu de traçabilitat detallada entre requisits funcionals i casos d'ús.

8.2.4 Diagrama de casos d'ús

A continuació es mostra el diagrama general de casos d'ús, que resumeix la interacció dels actors principals (Usuari i Administrador) amb les funcionalitats clau del sistema.

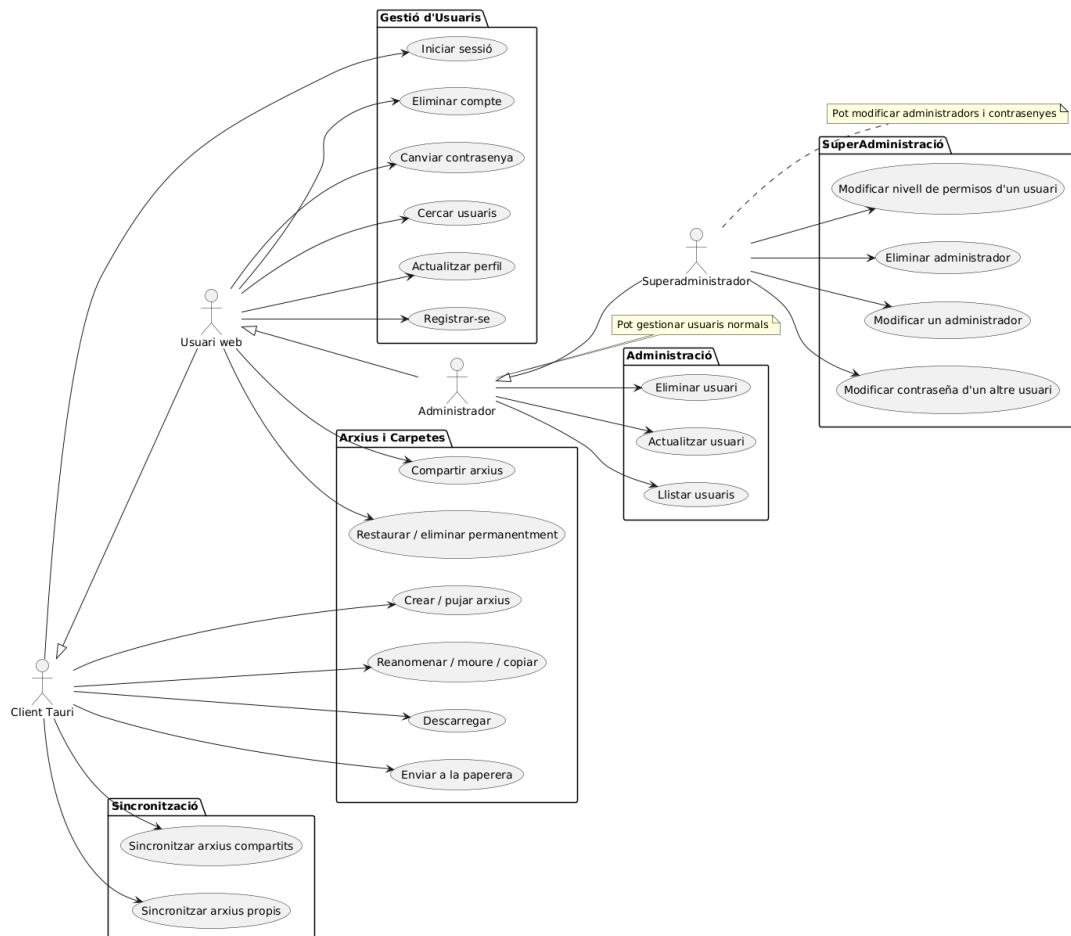


FIGURA 8.1: Diagrama de casos d'ús

Tal com mostra el diagrama, el sistema té diferents tipus d'usuaris (actors). El punt de partida és el **Client Tauri**, l'aplicació d'escritori pensada per a una integració total amb el sistema de fitxers local. L'**Usuari web** hereta d'aquest les funcions bàsiques de gestió d'arxius, com pujar-ne, descarregar-ne o compartir-los, però operant des del navegador.

El que els diferencia és la sincronització. Tot i que tots dos clients en tenen, la seva funció és distinta. El client **Tauri** busca la **rèplica de fitxers**, és a dir, que una carpeta local sigui un mirall del que hi ha al servidor. En canvi, a l'**aplicació web** la sincronització s'utilitza per refreshar l'estat de la interfície. D'aquesta manera, si un altre usuari, o el mateix usuari en un altre instantia de l'aplicació, fa un canvi, aquest es reflecteix a la pantalla al moment, sense haver de recarregar la pàgina. Cal dir que, tot i que el disseny inicial preveia la sincronització d'arxius compartits a Tauri, aquesta funció no es va poder implementar per falta de temps, com ja s'explica al capítol 4.

Per sobre del usuari normal, tenim l'**Administrador**. Aquest rol, a part de fer el mateix que un usuari normal, té la capacitat de gestionar altres comptes d'usuari: pot consultar la llista, modificar-ne les dades o directament eliminar-los.

Finalment, tenim el **Superadministrador**. És el rol amb més privilegis, ja que fa tot el que fan els altres i, a més, pot gestionar els comptes dels propis administradors, i també la capacitat de modificar les contrasenyes d'altres usuaris (independentment del seu rol) o modificar el rol d'un altre usuari. Això garanteix un control absolut sobre el sistema.

El disseny d'aquesta jerarquia respon a una filosofia de control centralitzat. La figura del **Superadministrador** es concep com un rol únic, amb autoritat sobre tot el sistema, que es crea exclusivament durant el procés d'instal·lació mitjançant scripts, no des de l'aplicació. Això garanteix un punt de control segur. A partir d'aquí, el Superadministrador pot delegar funcions nomenant altres **Administradors**, creant una estructura piramidal. No obstant això, les accions més sensibles —com modificar contrasenyes alienes o canviar rols— queden reservades per a ell. Aquesta concentració de poder en una única figura el converteix en el màxim responsable de la seguretat i la integritat de la plataforma. Al ser un rol generat durant el procés d'instal·lació, també s'assumeig que sera el gestor principal de la aplicació (qui inicia el procés d'instal·lació) qui asumira aquest rol al ser també el major responsable del manteniment de les dades y la seguretat del sistema.

8.2.5 Diagrames d'activitat dels Casos d'ús Principals

A continuació, es presenten els diagrames d'activitat per als casos d'ús més representatius del sistema. Cada diagrama il·lustra el flux de treball, les decisions i les interaccions entre serveis.

UC-01: Registrar-se

El flux comença quan l'usuari envia el formulari de registre. El Gateway reenvia la petició a UserAuthentication, que valida el format de les dades, comprova que el nom d'usuari no existeix i consulta a UserManagement per assegurar que l'email tampoc estigui en ús. Si tot és correcte, orquestra la creació de l'usuari: guarda credencials, demana a UserManagement que creï el perfil, a FileManagement que generi la carpeta arrel i notifica a SyncService via RabbitMQ. Finalment, retorna els tokens per iniciar la sessió.

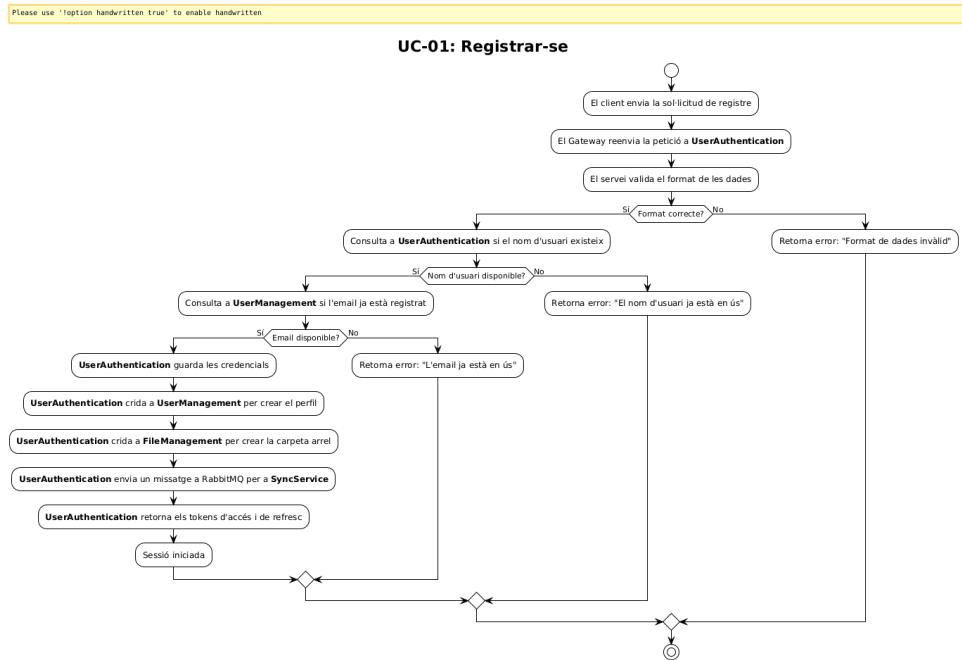


FIGURA 8.2: Diagrama d'activitat per al cas d'ús UC-01: Registrar-se.

UC-02: Iniciar sessió

L'usuari envia les seves credencials, que el Gateway reenvia a UserAuthentication. El servei busca l'usuari i, si existeix, verifica la contrasenya. Si les credencials són correctes, genera i retorna un nou joc de tokens (accés i refresh) per activar la sessió del client.

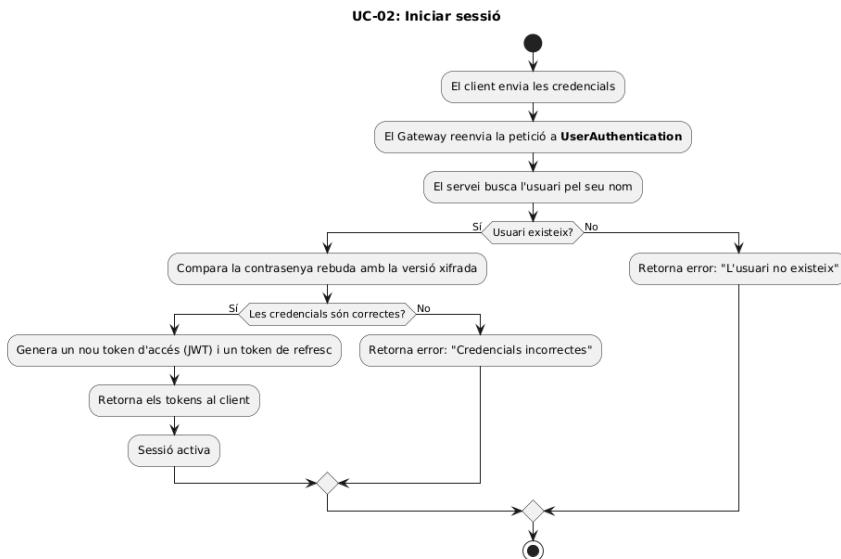


FIGURA 8.3: Diagrama d'activitat per al cas d'ús UC-02: Iniciar sessió.

UC-08: Crear o pujar arxius

FileManagement rep la petició i consulta a FileAccessControl si l'usuari té permís d'escriptura. Si és així, crea les metadades, emmagatzema el fitxer (si escau), demana a FileAccessControl que assigni el permís de propietari i finalment notifica SyncService del canvi.

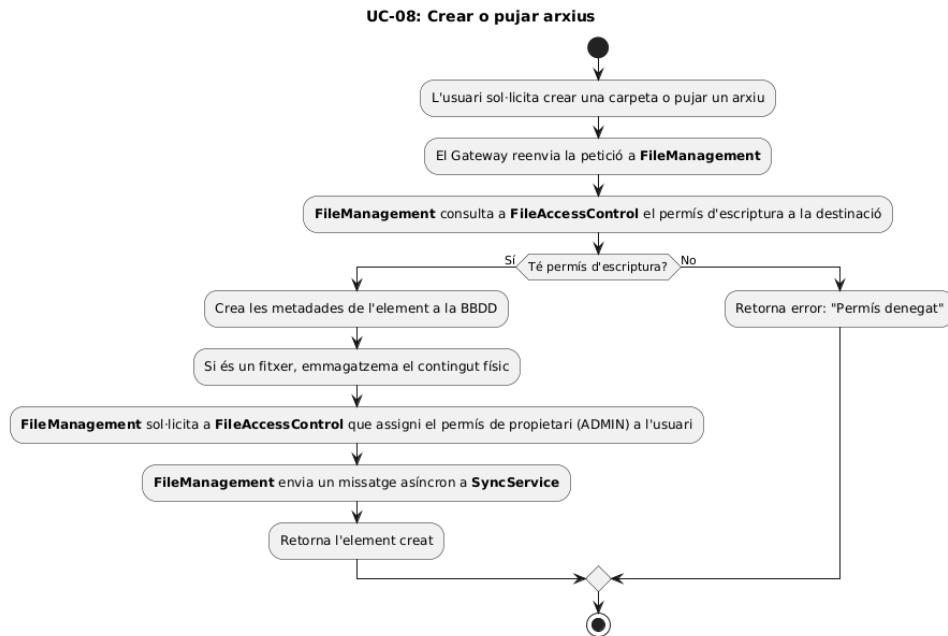


FIGURA 8.4: Diagrama d'activitat per al cas d'ús UC-08: Crear o pujar arxius.

UC-10: Descarregar

Després de verificar el permís de lectura a FileAccessControl, FileManagement recupera el fitxer del sistema d'emmagatzematge (o el comprimeix si és una carpeta) i el retorna al client com un flux de dades.

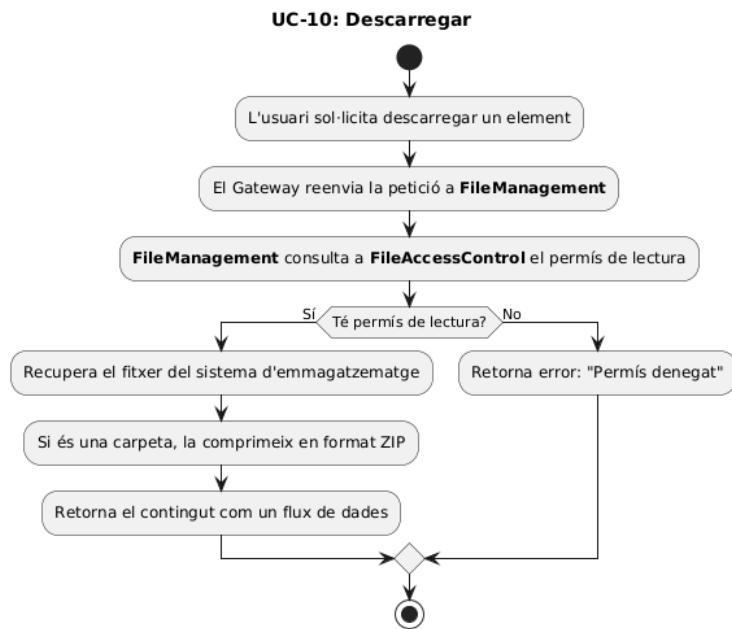


FIGURA 8.5: Diagrama d'activitat per al cas d'ús UC-10: Descarregar.

UC-11: Enviar a la paperera

TrashService rep la petició, verifica el permís d'escriptura a FileAccessControl i demana a FileManagement que marqui l'element com a eliminat. Finalment, crea un registre a la seva pròpia base de dades per gestionar la caducitat de l'element.

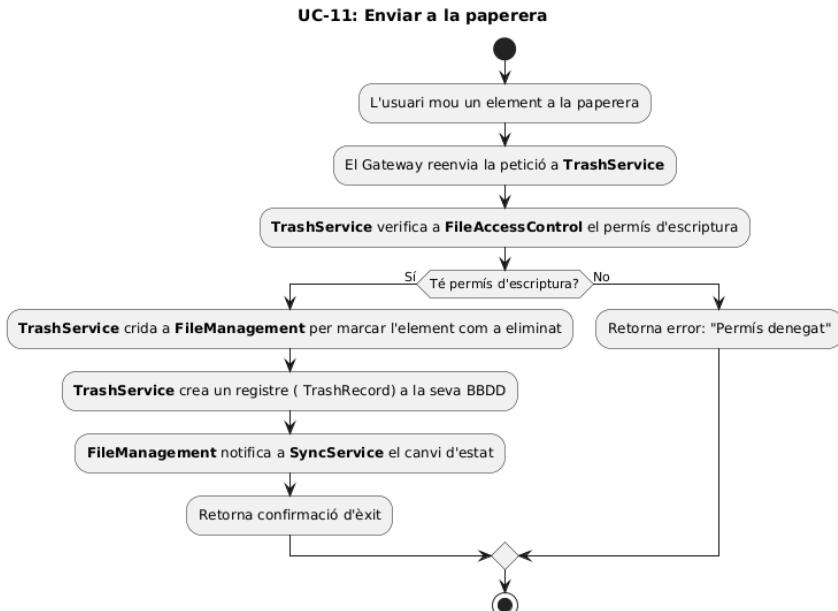


FIGURA 8.6: Diagrama d'activitat per al cas d'ús UC-11: Enviar a la paperera.

UC-12: Eliminar permanentment

Quan un usuari sol·licita l'eliminació permanent, TrashService verifica que n'és el propietari. Si ho és, inicia la saga d'eliminació enviant missatges a la cua perquè FileManagement, FileAccessControl i FileSharing purguin totes les dades associades de forma asíncrona.

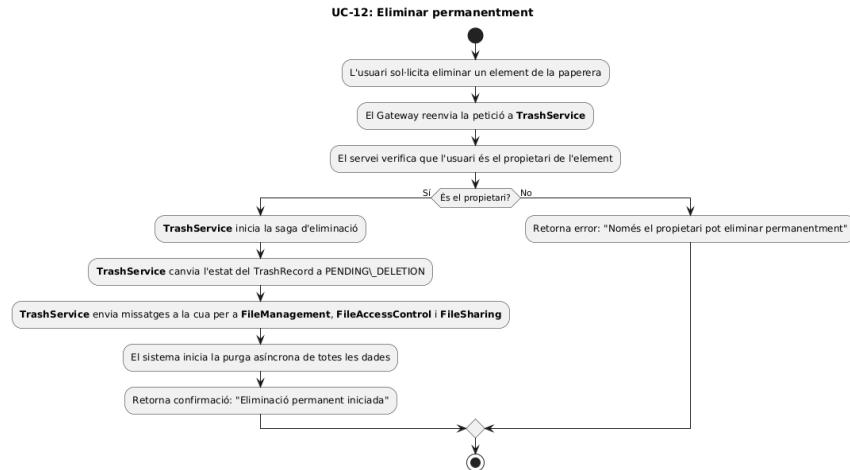


FIGURA 8.7: Diagrama d'activitat per al cas d'ús UC-12: Eliminar permanentment.

UC-13: Compartir arxius

El servei FileSharing comprova que el sol·licitant és el propietari de l'element. Després, obté l'ID de l'usuari convidat de UserManagement i demana a FileAccessControl que creï la nova regla d'accés. Finalment, desa un registre de la compartició i notifica SyncService.

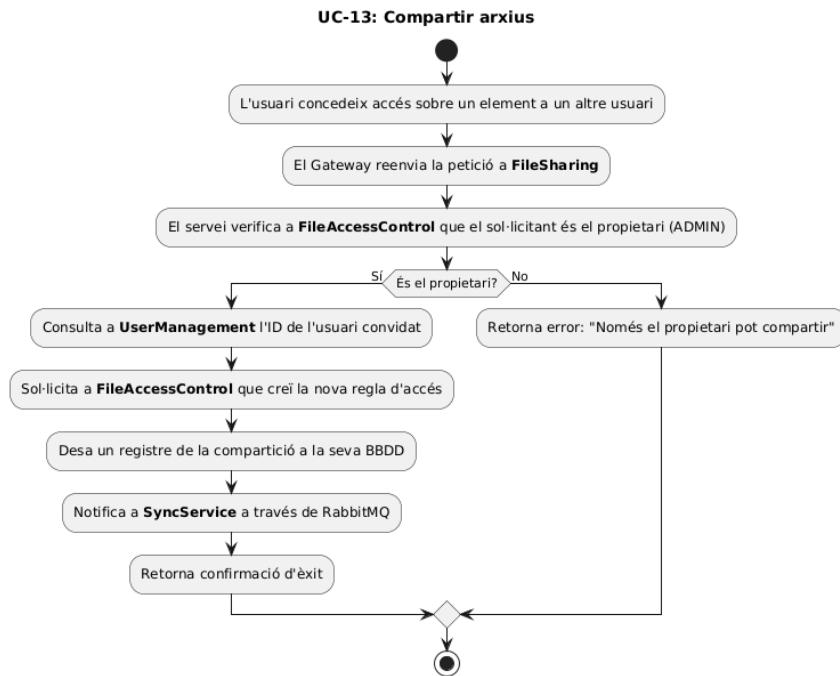


FIGURA 8.8: Diagrama d'activitat per al cas d'ús UC-13: Compartir arxius.

(La resta de diagrames d'activitat detallats per a cada cas d'ús es poden consultar a l'Apèndix B).

8.3 Arquitectura del sistema

8.3.1 Visió general i components

El sistema segueix una arquitectura de microserveis, dissenyada per garantir l'escalabilitat, la resiliència i la mantenibilitat. La figura 8.9 mostra una visió general d'aquesta arquitectura, els components principals de la qual es descriuen a continuació.

- **Gateway:** Actua com a punt d'entrada únic (*Single Point of Entry*) per a totes les sol·licituds dels clients. La seva funció és enrutar les peticions REST a l'API cap al microservei corresponent i gestionar les connexions WebSocket per a les actualitzacions en temps real, dirigint-les exclusivament al **SyncService**. Aquesta capa d'abstracció simplifica la comunicació des del client i centralitza la gestió de l'autenticació i el control d'accés inicial.
- **Core Services:** Constitueixen el nucli funcional de l'aplicació. Estan agrupats en dos paquets lògics:
 - **User & Auth:** Conté els serveis responsables de l'autenticació (UserAuthentication) i la gestió de les dades dels usuaris (UserManagement).
 - **File System:** Agrupa tots els serveis relacionats amb la gestió d'arxius, incloent la manipulació de metadades (FileManagement), el control d'accés

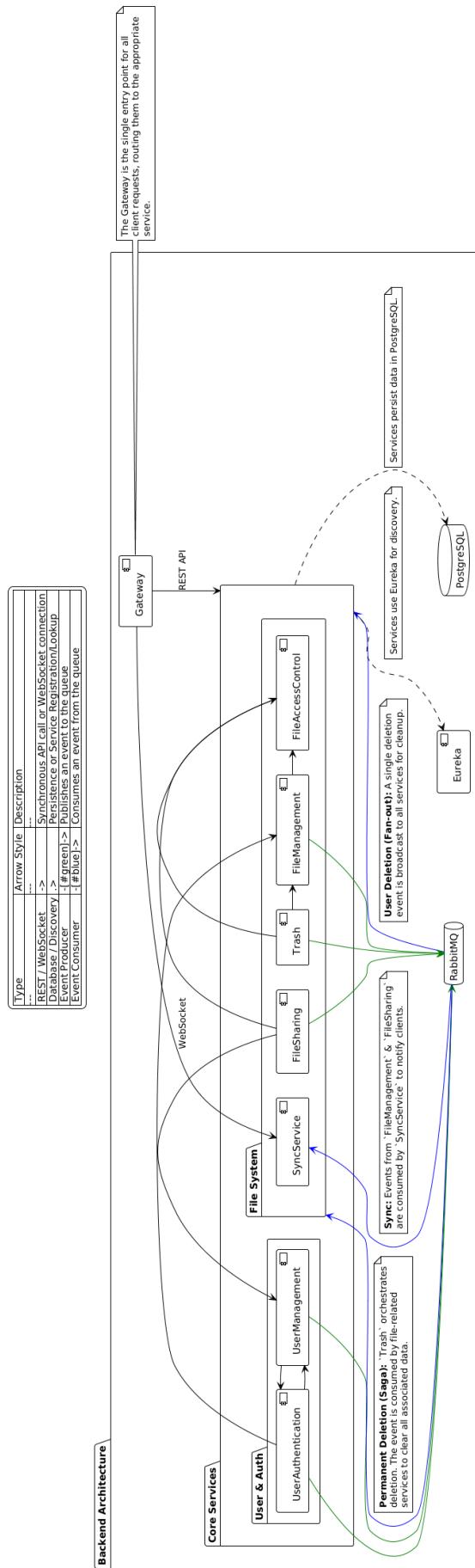


FIGURA 8.9: Diagrama de componentes de l'arquitectura del backend.

(FileAccessControl), la compartició (FileSharing), la gestió de la paperera (Trash) i la sincronització en temps real (SyncService).

- **Components d'Infraestructura:**

- **Eureka Server:** Actua com a registre de serveis. Cada microservei es registra a Eureka en iniciar-se, la qual cosa permet el descobriment dinàmic de serveis dins de la xarxa interna.

- **PostgreSQL Database:** És el sistema de gestió de bases de dades relacional on tots els serveis principals persisteixen les seves dades. Encara que comparteixen la mateixa instància de base de dades, cada servei opera sobre el seu propi esquema per mantenir un acoblament baix.

- **Comunicació Asíncrona (RabbitMQ):** Per a operacions que requereixen un alt grau de desacoblament o que són de llarga durada, s'utilitza una cua de missatges amb RabbitMQ. Els principals fluxos asíncrons són:

- **Sincronització en Temps Real:** Serveis com FileManagement i FileSharing publiquen esdeveniments (p. ex., creació o modificació d'un arxiu). SyncService consumeix aquests esdeveniments per notificar els clients connectats via WebSocket.

- **Eliminació d'Usuari (Fan-out):** Quan s'inicia l'eliminació d'un usuari, es publica un únic missatge que és rebut per tots els serveis. Aquest patró (*fan-out*) assegura que cada servei pugui purgar de forma independent totes les dades associades a l'usuari eliminat.

- **Eliminació Permanent (Saga):** El servei Trash orquestra l'eliminació definitiva d'un fitxer mitjançant una saga. Aquest patró s'utilitza perquè l'eliminació no és una acció atòmica, sinó una transacció distribuïda que ha d'executar-se de forma coordinada en diversos serveis. La saga, implementada mitjançant missatges, garanteix que l'operació es compleix de forma resilient: Trash publica un missatge que és consumit pels serveis del paquet *File System* per garantir que es netegen totes les dades relacionades (metadades, permisos i comparticions) de forma eventualment consistent.

Aquesta estructura modular permet un desenvolupament i desplegament independents de cada component. A continuació, es detalla el disseny específic de cada microservei, incloent les seves responsabilitats, el seu diagrama de classes i l'esquema de la seva base de dades, per proporcionar una visió completa del seu funcionament intern.

8.3.2 Disseny dels microserveis

UserAuthentication

Aquest servei és el responsable de gestionar les credencials dels usuaris (nom d'usuari, contrasenya) i els seus rols. Centralitza els processos de registre, inici de sessió i validació de tokens JWT.

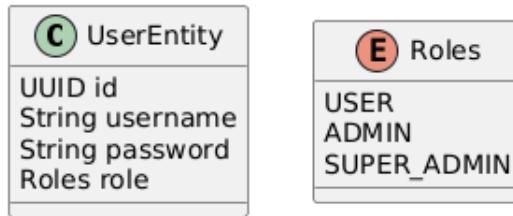


FIGURA 8.10: Diagrama de classes del servei UserAuthentication.

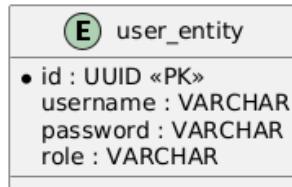


FIGURA 8.11: Diagrama de la base de dades del servei UserAuthentication.

Per optimitzar el rendiment de les consultes, s'ha creat un índex a la columna email de la taula user_info. Aquest índex és crucial per accelerar la comprovació d'existeància de correus durant el registre d'usuaris i per a les cerques ràpides basades en l'adreça de correu electrònic.

UserManagement

Gestiona la informació personal dels usuaris, com el correu electrònic, el nom i els cognoms. Col·labora estretament amb UserAuthentication durant el registre i proporciona funcionalitats per a la cerca i gestió d'usuaris per part dels administradors.

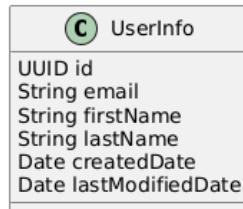


FIGURA 8.12: Diagrama de classes del servei UserManagement.

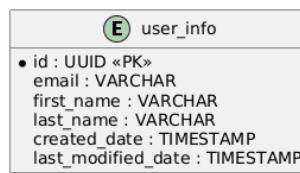


FIGURA 8.13: Diagrama de la base de dades del servei UserManagement.

Per optimitzar el rendiment de les consultes, s'ha creat un índex a la columna email de la taula user_info. Aquest índex és crucial per accelerar la comprovació d'existeència de correus durant el registre d'usuaris i per a les cerques ràpides basades en l'adreça de correu electrònic.

FileManagement

És el nucli del sistema de gestió de fitxers. S'encarrega de les metadades dels arxius i carpetes (nom, mida, dates) i de la seva ubicació física al servidor. Processa operacions com la creació, pujada, descàrrega, renombrat i moviment d'elements.

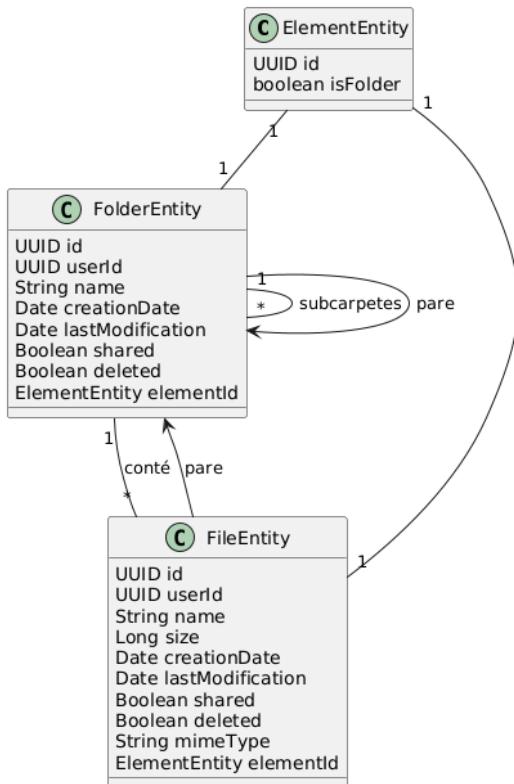


FIGURA 8.14: Diagrama de classes del servei FileManagement.

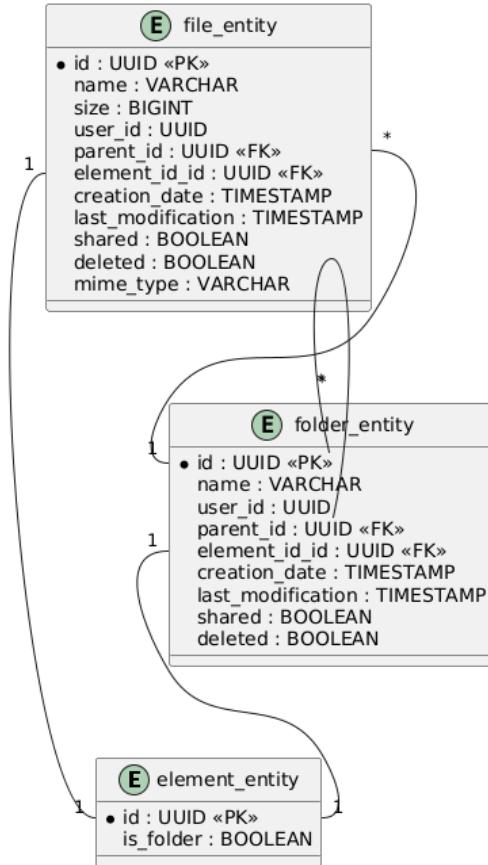


FIGURA 8.15: Diagrama de la base de dades del servei FileManager.

Per garantir un rendiment òptim, s'han definit diversos índexs. A les taules **file_entity** i **folder_entity**, s'han indexat les columnes **user_id** i **parent_id**. L'índex sobre **user_id** accelera la càrrega inicial dels arxius d'un usuari, mentre que l'índex sobre **parent_id** és fonamental per llistar de forma ràpida el contingut d'una carpeta, una de les operacions més freqüents del sistema.

FileAccessControl

Aquest servei actua com a autoritat central per a la gestió de permisos. Emmagatzema les regles que defineixen quin usuari té quin tipus d'accés (lectura, escriptura, propietari) sobre cada fitxer o carpeta. És consultat per altres serveis abans de realitzar qualsevol operació crítica.

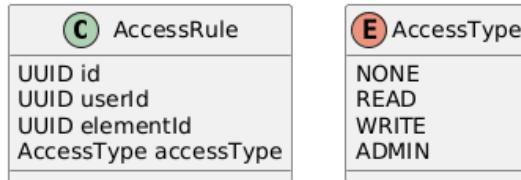


FIGURA 8.16: Diagrama de classes del servei FileAccessControl.

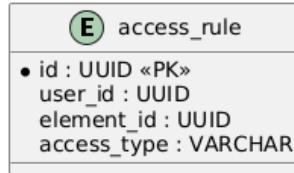


FIGURA 8.17: Diagrama de la base de dades del servei FileAccess-Control.

La taula `access_rule` està fortament indexada per les columnes `user_id` i `element_id`. Aquests índexs són essencials per a la funció principal del servei: verificar de manera quasi instantània si un usuari concret té permís sobre un element específic, una consulta que es realitza abans de la majoria d'operacions sobre fitxers.

FileSharing

Gestiona la lògica de compartició d'arxius entre usuaris. Emmagatzema registres de qui ha compartit què i amb qui, i col·labora amb FileAccessControl per aplicar els permisos corresponents als usuaris convidats.

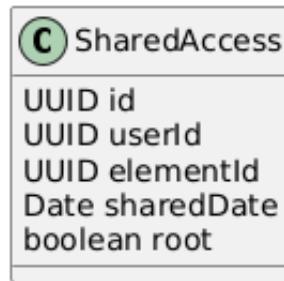


FIGURA 8.18: Diagrama de classes del servei FileSharing.

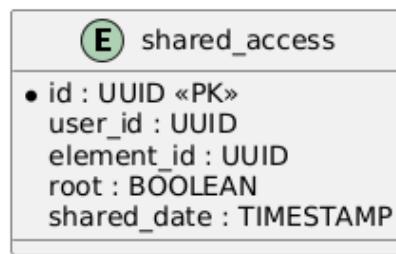


FIGURA 8.19: Diagrama de la base de dades del servei FileSharing.

Per optimitzar les consultes relacionades amb la compartició, s'han creat índexs a les columnes `user_id` i `element_id` de la taula `shared_access`. Aquests permeten recuperar ràpidament tots els elements compartits amb un usuari o, inversament, tots els usuaris amb qui s'ha compartit un element.

Trash

Implementa la funcionalitat de la paperera de reciclatge. Quan un usuari elimina un element, aquest servei el marca com a "eliminat" emmagatzema un registre amb la data de caducitat. També orquestra el procés d'eliminació permanent (la saga descrita anteriorment).

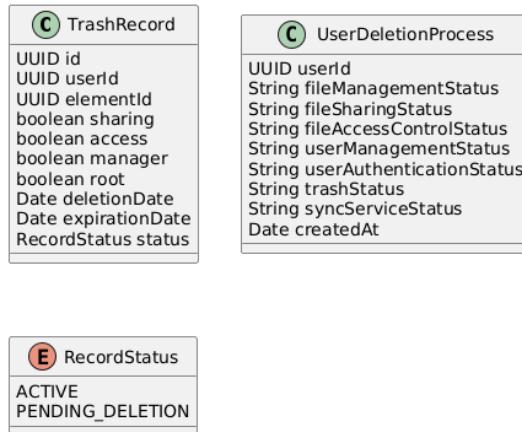


FIGURA 8.20: Diagrama de classes del servei Trash.

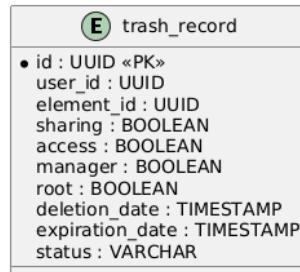


FIGURA 8.21: Diagrama de la base de dades del servei Trash.

El rendiment de la paperera es millora amb índexs a les columnes `user_id` i `element_id`. El primer accelera la càrrega de la vista de la paperera per a un usuari, mentre que el segon permet localitzar de forma eficient el registre d'un element concret quan es vol restaurar o eliminar.

SyncService

És el responsable de les actualitzacions en temps real. Manté una connexió Web-Socket amb els clients actius i consumeix esdeveniments de RabbitMQ per notificar canvis en l'estructura de fitxers, mantenint així les interfícies d'usuari sincronitzades.

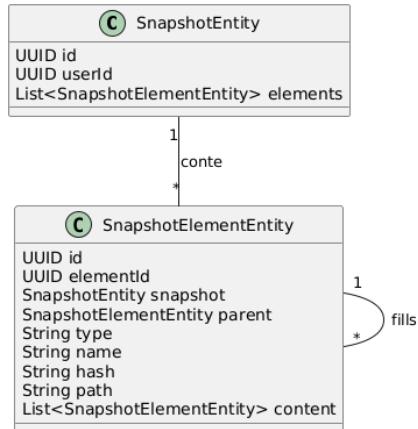


FIGURA 8.22: Diagrama de classes del servei SyncService.

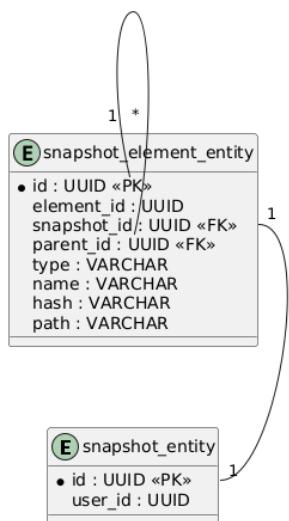


FIGURA 8.23: Diagrama de la base de dades del servei SyncService.

Per assegurar la rapidesa en les operacions de sincronització, s'han establert índexs clau. A la taula snapshot_entity, s'indexa user_id per localitzar ràpidament l'estat de sincronització d'un usuari. A la taula snapshot_element_entity, s'indexen snapshot_id per carregar tots els elements d'una sincronització i element_id per trobar un fitxer o carpeta específic dins de l'estructura de sincronització.

8.3.3 Patrons de disseny i principis arquitectònics

El desenvolupament del backend s'ha guiat per un conjunt de patrons de disseny i principis arquitectònics orientats a garantir un codi net, mantenible, escalable i desacoblat. L'ús del framework Spring facilita l'adopció d'aquests patrons, especialment a través del seu contenidor d'Inversió de Control (IoC) i la injecció de dependències.

- **Arquitectura per Capes:** Tots els microserveis segueixen una arquitectura de tres capes ben diferenciades (Controlador, Servei i Repositori). Aquesta separació de responsabilitats és fonamental:

- La **Capa de Controlador** gestiona les peticions HTTP i actua com a façana de l'API.
- La **Capa de Servei** conté la lògica de negoci del domini.
- La **Capa de Repositori** abstrau l'accés a les dades.

Aquesta estructura compleix el **Príncipi de Responsabilitat Única (SRP)** de SOLID, ja que cada capa té un propòsit clar i aïllat, la qual cosa millora la cohesió i redueix l'acoblament.

- **Injecció de Dependències i Inversió de Control (IoC):** En lloc que els components creïn les seves pròpies dependències, el contenidor de Spring les "injecta" automàticament. Per exemple, un servei no crea la seva instància de repositori, sinó que la declara com una dependència. Això compleix el **Príncipi d'Inversió de Dependències (DIP)** de SOLID, ja que els mòduls d'alt nivell (serveis) depenen d'abstraccions (interfícies de repositori) en lloc d'implementacions concretes. El resultat és un sistema molt més modular i fàcil de provar, ja que les dependències es poden substituir per simulacres (*mocks*) en els tests unitaris.
- **Patró Repository:** Implementat a través de Spring Data JPA, aquest patró desacbla la lògica de negoci de la tecnologia de persistència de dades. Els serveis interactuen amb una interfície (p. ex., UserRepository) sense conèixer els detalls de la base de dades subjacentes. Això permetria, per exemple, canviar de PostgreSQL a una altra base de dades SQL amb un impacte mínim en el codi de l'aplicació.
- **Data Transfer Object (DTO):** El sistema utilitza objectes específics per a les peticions ('...Request') i respuestes ('...Response') de l'API. Aquest patró desacbla el model de dades intern (entitats JPA) del model exposat públicament. Això no només és una bona pràctica per a la seguretat, evitant l'exposició excessiva de dades, sinó que també permet que l'API evolucioni de forma independent al model de la base de dades.

A més d'aquests principis generals, en el codi dels microserveis s'han implementat patrons de disseny específics per resoldre problemes concrets del domini:

- **Patró Composite:** En el servei FileManagement, les entitats ElementEntity, FileEntity i FolderEntity implementen aquest patró. Permet tractar de manera uniforme els objectes individuals (fitxers) i els compostos (carpetes), simplificant enormement les operacions recursives com moure o eliminar una carpeta amb tot el seu contingut.
- **Patró State:** El servei Trash utilitza una implementació d'aquest patró. L'entitat TrashRecord té un camp d'estat (RecordStatus) que determina el seu comportament: un element només es pot restaurar si el seu estat és ACTIVE, però no si està PENDING_DELETION.
- **Patró Factory Method (simplificat):** A UserAuthentication, el mètode generateToken actua com una fàbrica. El client sol·licita un token, i el mètode, basant-se en un paràmetre, decideix si ha de crear un token d'accés (de curta durada) o un de refresh (de llarga durada), encapsulant-ne la lògica de creació.

- **Patró d'Agregació:** El disseny de les dades d'usuari segueix aquest principi. En lloc de tenir una entitat monolítica, la informació s'ha separat en dos microserveis:
 - UserAuthentication: Gestiona les dades crítiques de seguretat (credencials, rol).
 - UserManagement: Gestiona la informació personal (nom, correu electrònic).

Aquesta agregació, connectada per un userId comú, permet tractar l'usuari com una unitat lògica alhora que es mantenen les seves dades desacoblades i segures.

8.3.4 Justificació de l'arquitectura de microserveis

L'elecció d'una arquitectura de microserveis per a aquest projecte no va ser una decisió trivial, sinó una resposta estratègica als requisits funcionals i no funcionals del sistema, com l'escalabilitat, la resiliència i la mantenibilitat a llarg termini. A continuació, es justifiquen els motius principals d'aquesta elecció enfront d'una alternativa monolítica:

- **Escalabilitat independent:** En una aplicació de gestió de fitxers, no tots els components tenen la mateixa càrrega de treball. Per exemple, el servei FileManagement (operacions de fitxers) i el SyncService (connexions WebSocket) són susceptibles de rebre una càrrega molt més alta que el UserManagement. L'arquitectura de microserveis permet escalar horitzontalment només aquells serveis que ho necessiten, optimitzant l'ús de recursos sense haver de replicar tota l'aplicació.
- **Mantenibilitat i Cohesió:** El sistema es descompon en serveis petits i cohesionats, cadascun amb una única responsabilitat ben definida (p. ex., autenticació, gestió d'arxius, paperera). Aquesta separació facilita enormement el desenvolupament i el manteniment. Un desenvolupador pot treballar en el servei de compartició (FileSharing) sense necessitat de comprendre les complexitats internes del sistema de sincronització, la qual cosa redueix la càrrega cognitiva i accelera el cicle de desenvolupament.
- **Aïllament de fallades i resiliència:** En un sistema monolític, un error no controlat en una part del codi pot provocar la caiguda de tota l'aplicació. En canvi, amb microserveis, una fallada en un servei no crític (com podria ser el TrashService) no hauria d'affectar el funcionament de la resta del sistema, com l'autenticació o la gestió de fitxers. L'ús d'un registre de serveis com Eureka contribueix a aquesta resiliència, permetent que els serveis es descobreixin i es comuniquin de manera dinàmica fins i tot si algunes instàncies fallen.
- **Flexibilitat tecnològica:** Tot i que actualment tots els serveis estan desenvolupats amb Spring Boot, l'arquitectura de microserveis ofereix la llibertat d'adoptar diferents tecnologies per a diferents serveis en el futur. Si, per exemple, es descobrís que un servei de processament de fitxers requereix un llenguatge optimitzat per a computació intensiva, es podria desenvolupar i integrar sense afectar la resta de l'ecosistema tecnològic.
- **Desplegament independent i agilitat:** Cada microservei es pot desplegar de

forma autònoma. Això significa que una actualització al servei d'usuaris (UserManagement) es pot llançar a producció sense necessitat de tornar a provar i desplegar tot el sistema. Aquest cicle de desplegament més ràpid i menys arriscat augmenta l'agilitat del projecte i facilita la integració contínua.

En conclusió, l'arquitectura de microserveis proporciona la base per a un sistema robust, flexible i preparat per créixer, alineant-se perfectament amb els objectius d'un servei al núvol modern i escalable.

8.4 Disseny de les interfícies d'usuari

8.4.1 Client web

El disseny de la interfície d'usuari per al client web s'ha centrat a oferir una experiència intuïtiva i funcional, semblant a la d'altres serveis d'emmagatzematge al núvol. A continuació es descriuen les principals vistes i components.

Autenticació

El primer contacte de l'usuari amb l'aplicació són els formularis d'autenticació. Es presenta una finestra per a l'inici de sessió (Figura 8.24) i una altra per al registre de nous usuaris (Figura 8.25), amb validacions clares per a cada camp.

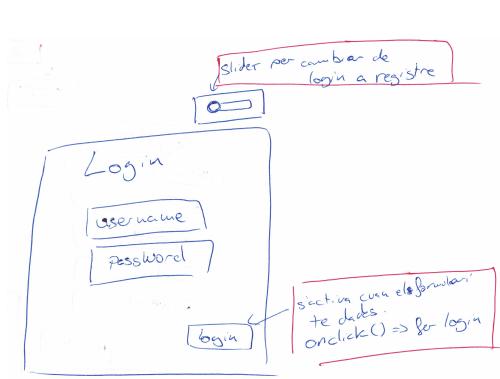


FIGURA 8.24: Pantalla d'inici de sessió.

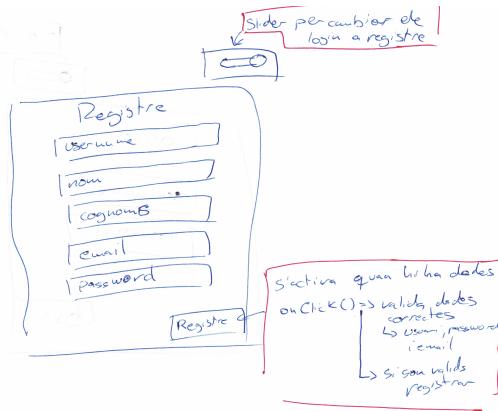


FIGURA 8.25: Pantalla de registre.

Escriptori principal

Un cop autenticat, l'usuari accedeix a l'escriptori principal (Figura 8.26), que és el nucli de l'aplicació. Aquesta vista es compon de diversos elements clau.

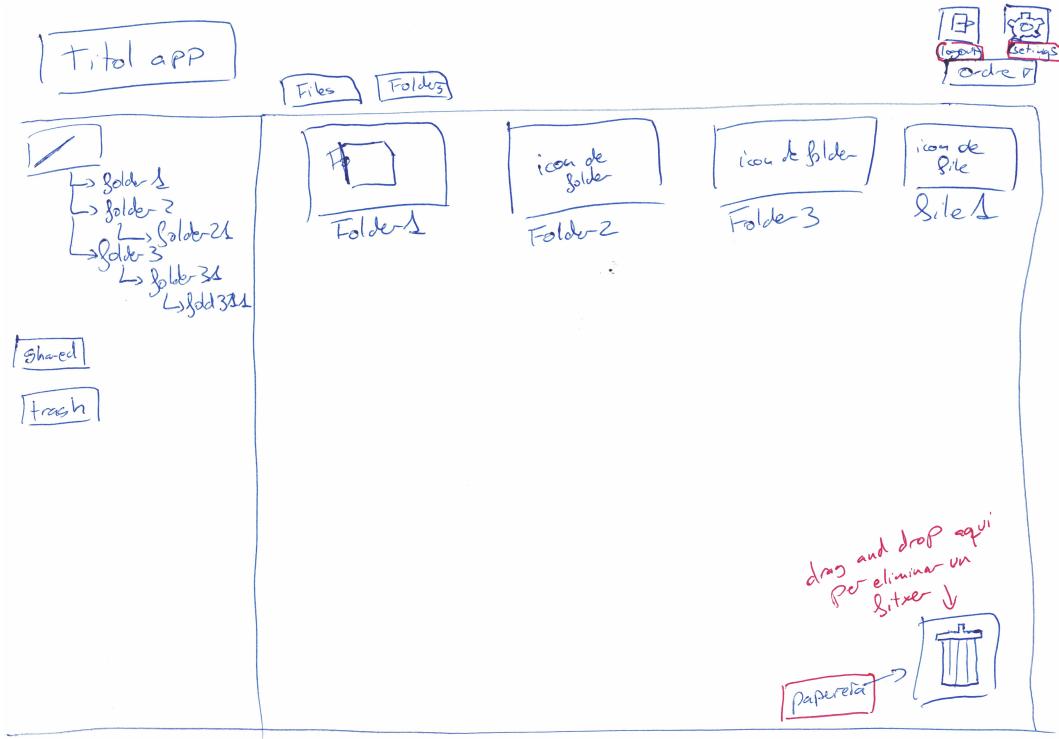


FIGURA 8.26: Escriptori principal del client web.

Navegació i accions principals A la part esquerra, un arbre de carpetes (Figura 8.27) permet navegar per l'estructura de directoris. En fer clic a una carpeta, s'accedeix al seu contingut i el seu estil canvia per reflectir que és la seleccionada. L'arbre inclou seccions per als arxius propis, els compartits i la paperera. La vista de contingut es pot filtrar per tipus d'element (fitxers o carpetes) mitjançant dos botons dedicats. Just a sobre d'aquest arbre, es troben els botons d'accio contextuals a la carpeta actual (Figura 8.28), que permeten pujar arxius, pujar carpetes senceres o crear una nova carpeta. Aquestes opcions també estan disponibles a la secció de 'Compartits amb mi', sempre que l'usuari tingui permisos d'escriptura a la carpeta seleccionada.

A la cantonada superior dreta (Figura 8.29), es troben les opcions globals: un menú per ordenar els fitxers (per nom, data, etc.), un botó per accedir al panell d'administració (només visible per a administradors), un altre per a la configuració del compte d'usuari, que obre un modal per modificar dades personals i canviar la contrasenya (Figura 8.30), i, finalment, el botó per tancar la sessió.

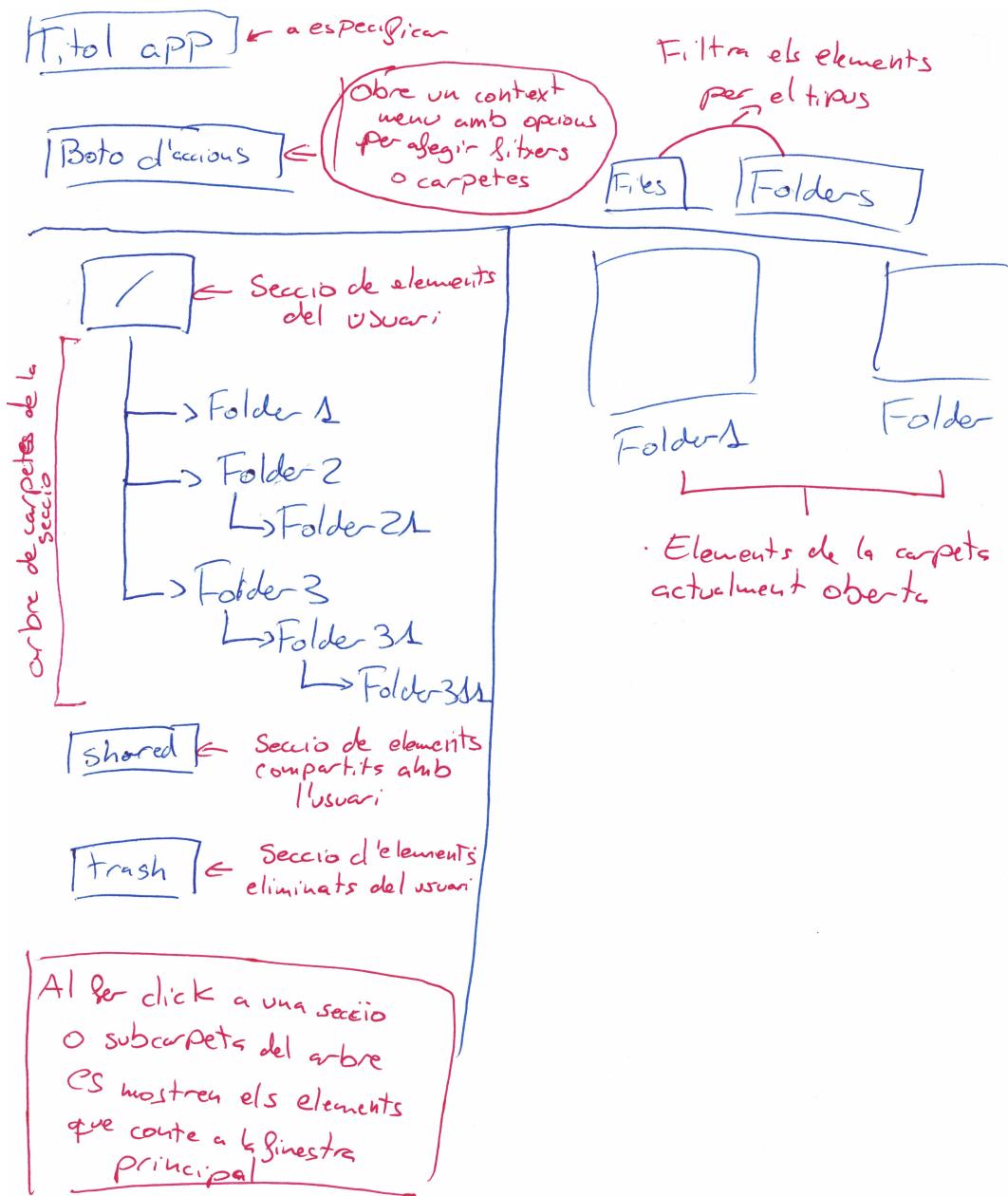


FIGURA 8.27: Detall de l'arbre de navegació, on s'indica la carpeta seleccionada i les diferents seccions.

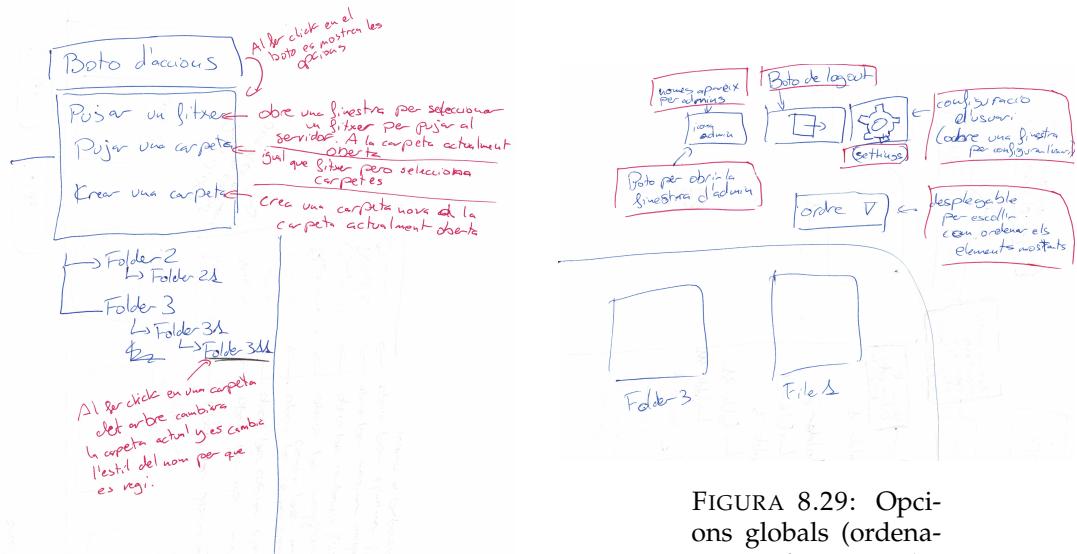


FIGURA 8.29: Opcions globals (ordenaçió, configuració i layout).

Configuració de l'usuari

FIGURA 8.30: Modal de configuració d'usuari.

Aquest modal permet modificar els dades d'un usuari:

- Campos** (modifiables):
 - Nom
 - Cognoms
 - Email
 - User name
 - Contraseña
- Botons**:
 - Cancelar**: Cancel·la l'operació.
 - Guardar**: Guarda els canvis.
- Textos descriptius**:
 - Tot modificable**: Indica que tots els camps són modificables.
 - Nones operatiu al modificar una data**: Indica que no hi ha operacions disponibles per modificar una data.
 - Al clickar es modifica l'usuari**: Indica que els canvis es reflecteixen immediatament en l'usuari.

Interacció amb elements Cada fitxer o carpeta disposa d'un menú contextual (Figura 8.31) que permet realitzar diverses operacions. Algunes d'aquestes accions es duen a terme a través de finestres modals dissenyades per a cada tasca específica:

- **Detalls:** Mostra informació rellevant de l'element (Figura 8.32).
- **Renombrar:** Permet canviar el nom de l'element (Figura 8.33).
- **Moure:** Facilita el trasllat d'elements a una altra carpeta (Figura 8.34).
- **Compartir:** Obre un modal on es pot buscar un usuari, assignar-li permisos i gestionar els accessos existents (Figura 8.35).

A més d'aquestes accions que es gestionen amb finestres modals, el menú contextual ofereix opcions d'acció directa com ara descarregar, copiar, tallar i eliminar (que mou l'element a la paperera).

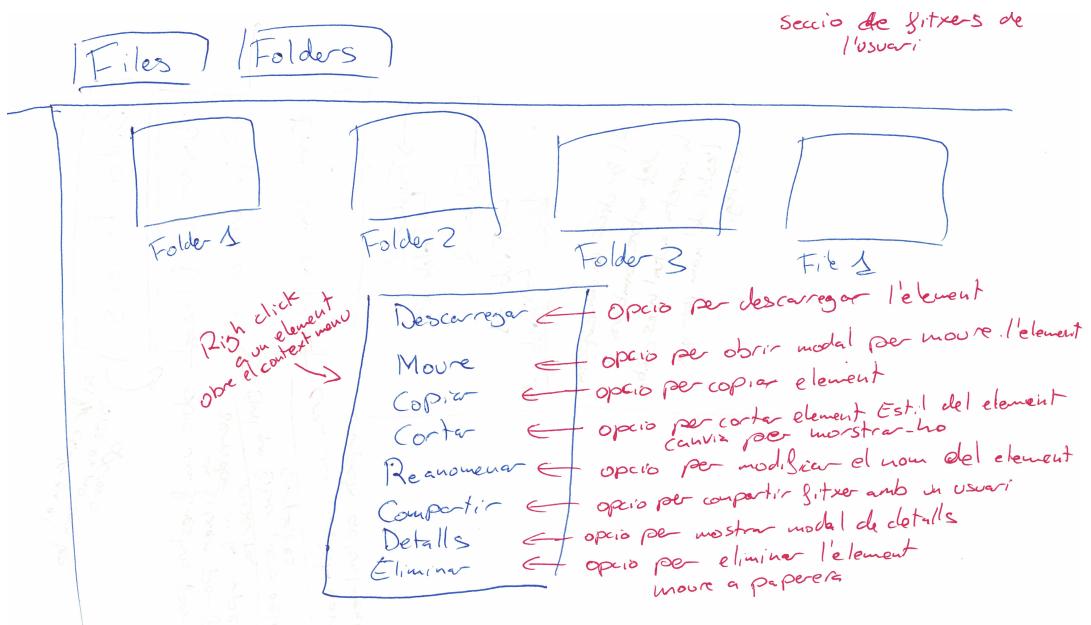


FIGURA 8.31: Menú contextual d'accions sobre un element.

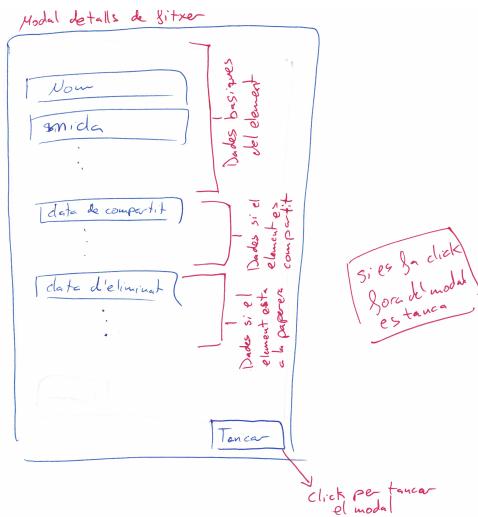


FIGURA 8.32: Modal amb els detalls d'un fitxer.

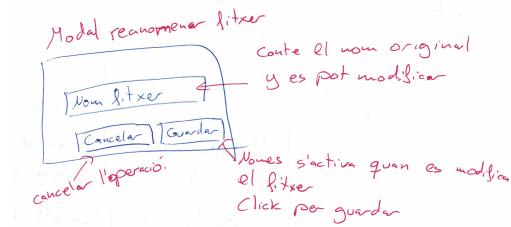


FIGURA 8.33: Modal per renombrar un element.

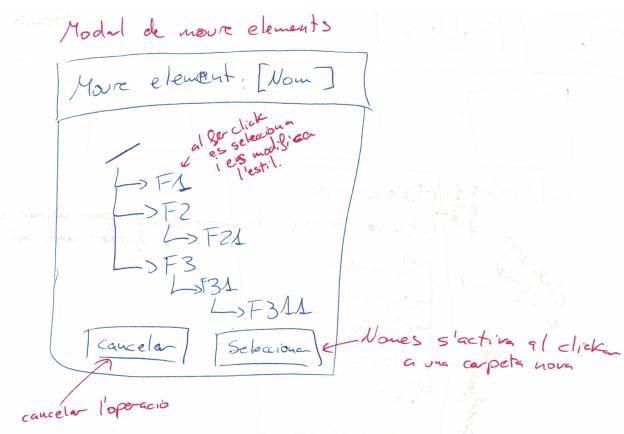


FIGURA 8.34: Modal per moure elements a una altra ubicació.

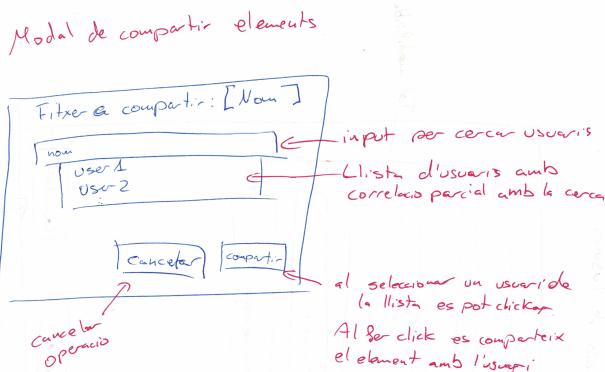


FIGURA 8.35: Modal de compartició d'arxius.

Gestió d'elements compartits Per als elements que es troben a la secció 'Compartits amb mi', les opcions del menú contextual són dinàmiques i depenen dels permisos

que l'usuari tingui sobre l'element (Figura 8.36). Les opcions només es mostren si l'usuari té els privilegis necessaris:

- Per a tots els fitxers compartits, existeix l'opció per deixar de compartir, que elimina l'accés de l'usuari a l'element.
- Si es tenen permisos de **lectura**, es pot descarregar l'arxiu i consultar-ne els detalls.
- Si es tenen permisos d'**escriptura**, a més de les anteriors, s'afegeixen les opcions de renombrar, copiar, tallar i moure l'element.

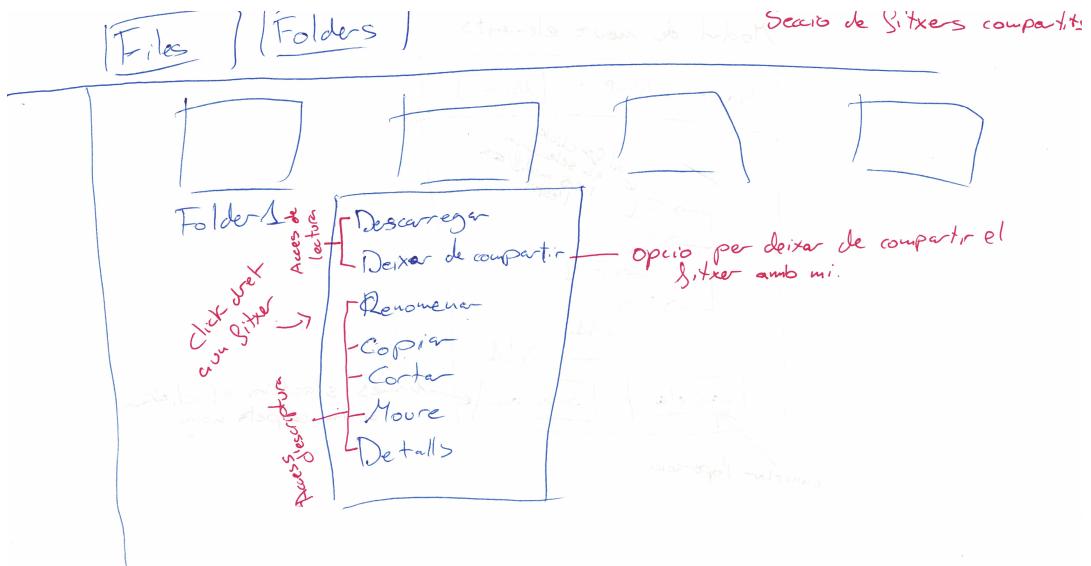


FIGURA 8.36: Menú contextual per a un element compartit.

Paperera La vista de la paperera (Figura 8.37) mostra tots els elements eliminats i ofereix un menú contextual amb diverses opcions per a cada un:

- **Restaurar:** Retorna l'element a la seva ubicació original.
- **Eliminar definitivament:** Esborra l'element de forma permanent del sistema.
- **Details:** Mostra la informació de l'arxiu.

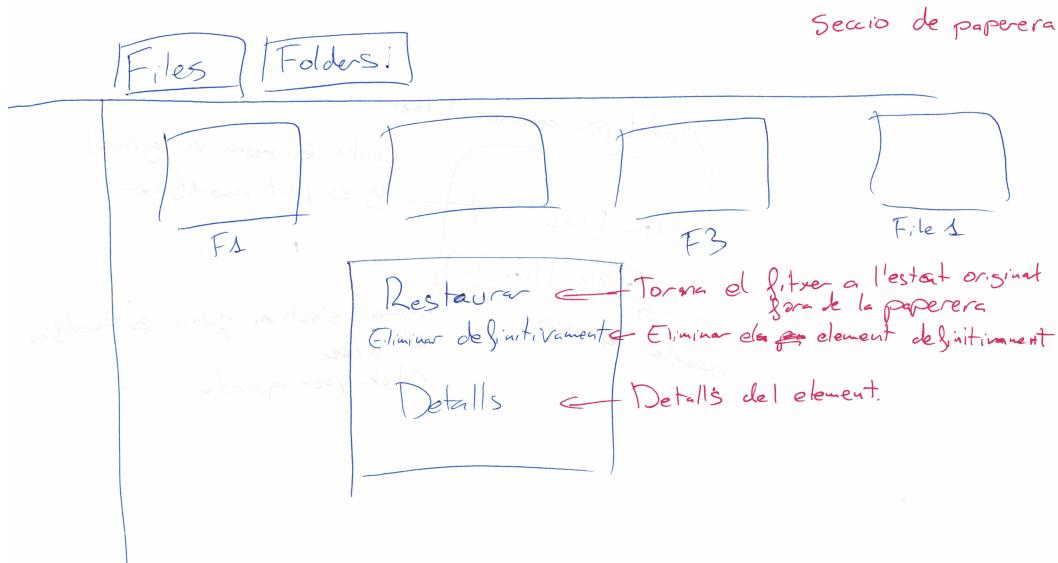


FIGURA 8.37: Opcions de la paperera.

Panell d'administració Finalment, els usuaris amb rol d'administrador tenen accés a un panell exclusiu (Figura 8.38) per gestionar els comptes d'usuari de la plataforma, on poden veure la llista d'usuaris, modificar-ne les dades o eliminar-los.

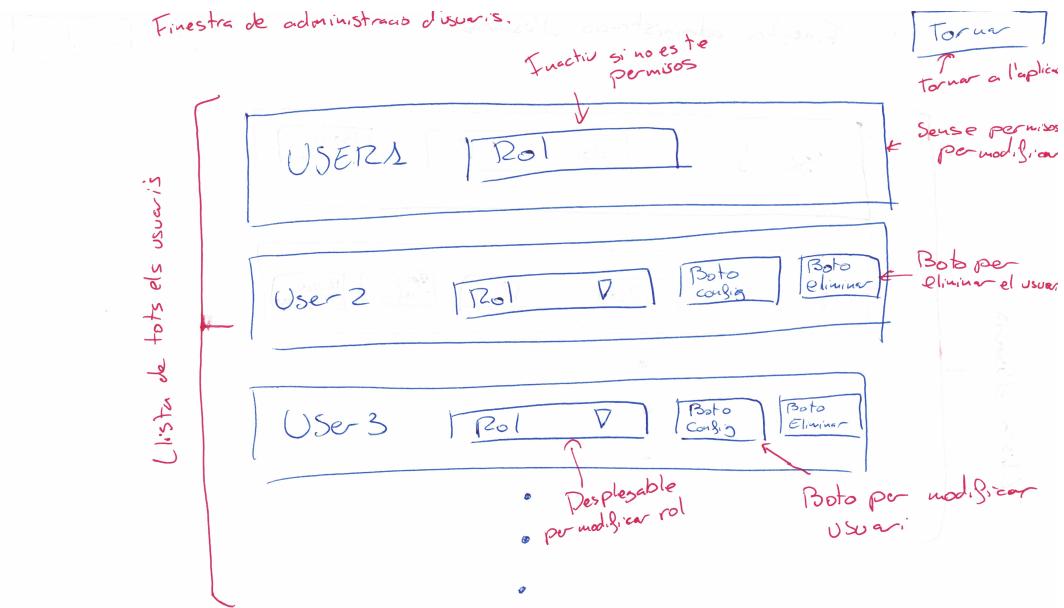


FIGURA 8.38: Panell d'administració d'usuaris.

8.4.2 Client d'escriptori

El disseny de l'aplicació d'escriptori amb Tauri s'ha centrat a oferir una experiència nativa i integrada amb el sistema operatiu, posant especial èmfasi en la sincronització automàtica de fitxers.

El primer cop que s'inicia l'aplicació, es presenta a l'usuari la finestra de configuració inicial (Figura 8.39). Aquí ha de definir dos paràmetres clau: l'endpoint del servidor

al qual es connectarà i la carpeta local que es mantindrà sincronitzada. Aquesta configuració és un pas previ indispensable per poder operar.

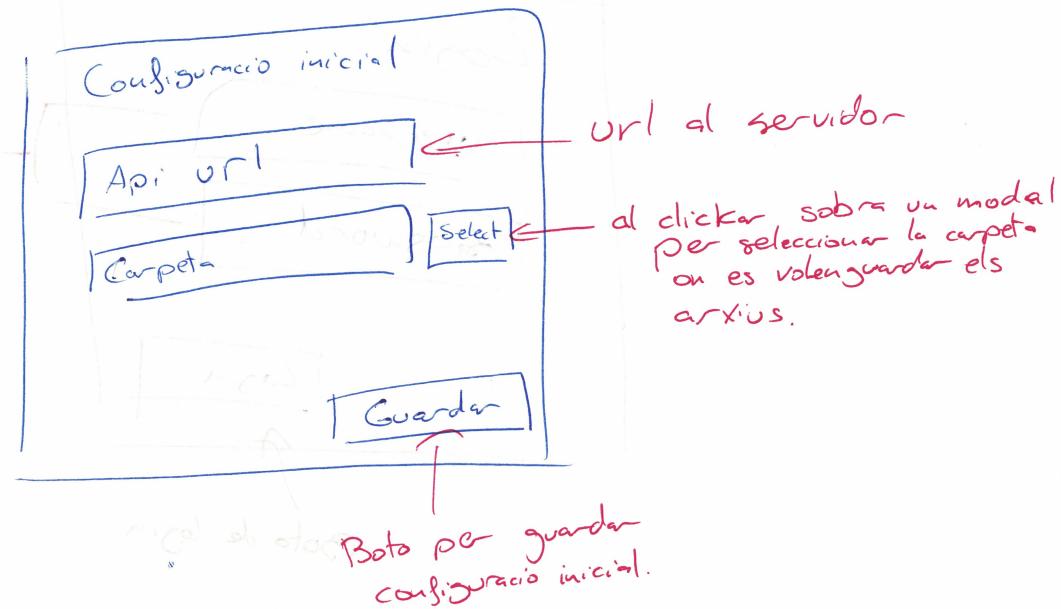


FIGURA 8.39: Configuració inicial del servidor i la carpeta de sincronització.

Un cop guardada la configuració, l'usuari ha d'iniciar sessió (Figura 8.40) per accedir al seu compte. L'aplicació desarà els tokens d'autenticació de manera segura, de manera que aquest pas només serà necessari la primera vegada o si la sessió caduca després d'un llarg període d'inactivitat.

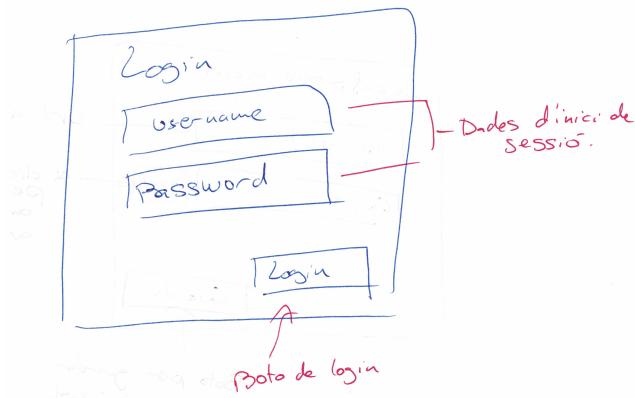


FIGURA 8.40: Finestra d'inici de sessió del client Tauri.

Després d'autenticar-se, l'usuari accedeix a les funcionalitats principals de l'aplicació:

- **Finestra principal de sincronització:** Aquesta és la vista principal de l'aplicació (Figura 8.41), on l'usuari pot monitorar l'estat de la sincronització. Mostra

un historial de les pujades i baixades actives i recents. A la part superior, disposa de quatre botons per a accions ràpides:

- Obrir la carpeta de sincronització local (icona de carpeta).
- Accedir a l’aplicació web (icona de globus terraquí).
- Forçar una sincronització manual (icona de dues fletxes circulars).
- Obrir el menú de configuració (icona d’engranatge).

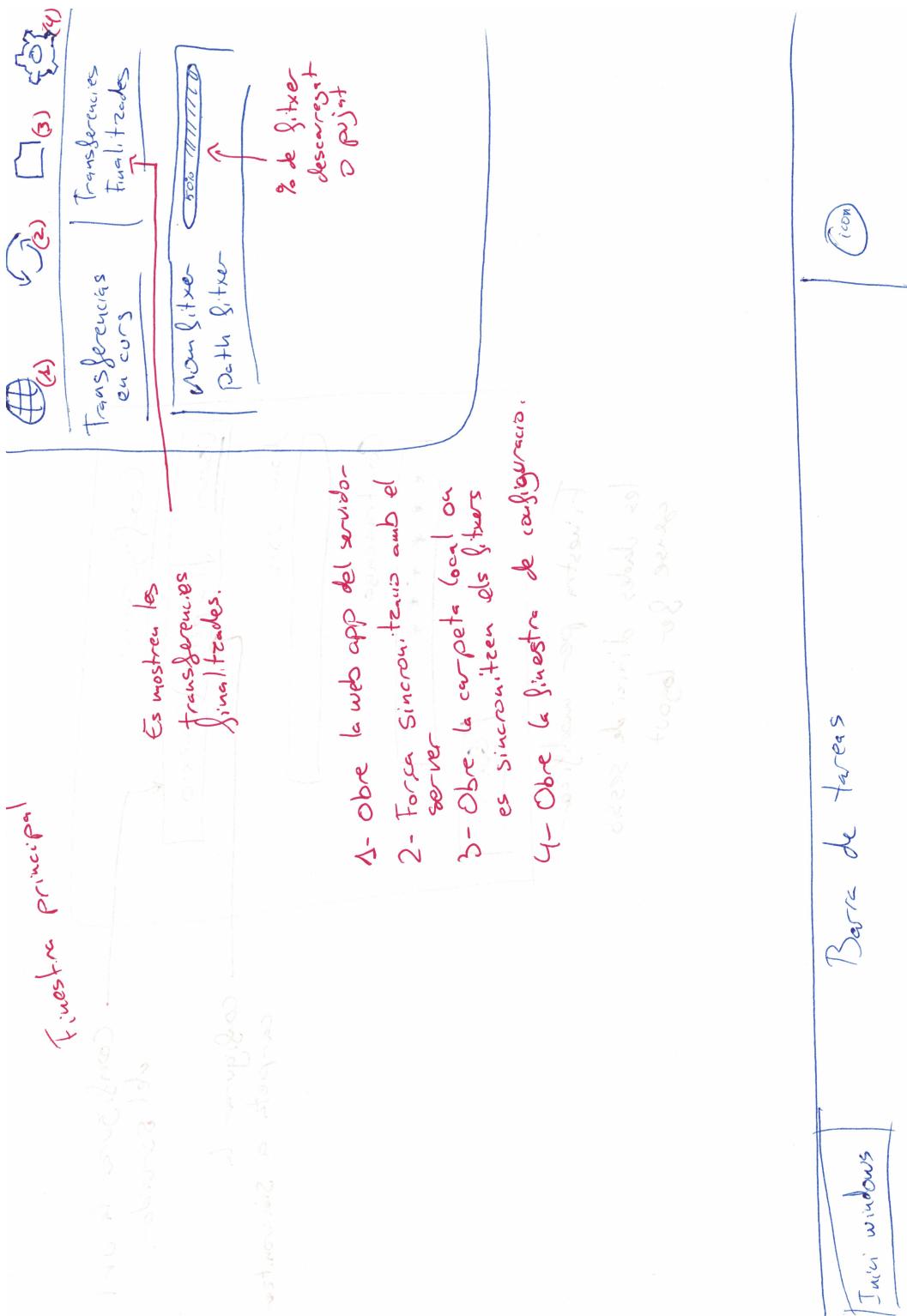


FIGURA 8.41: Finestra principal de sincronització del client d'escriptori.

- **Menú de configuració:** L'usuari pot accedir a una finestra de configuració (Figura 8.42) per ajustar paràmetres de l'aplicació, com les dades de l'usuari o canviar la carpeta de sincronització.

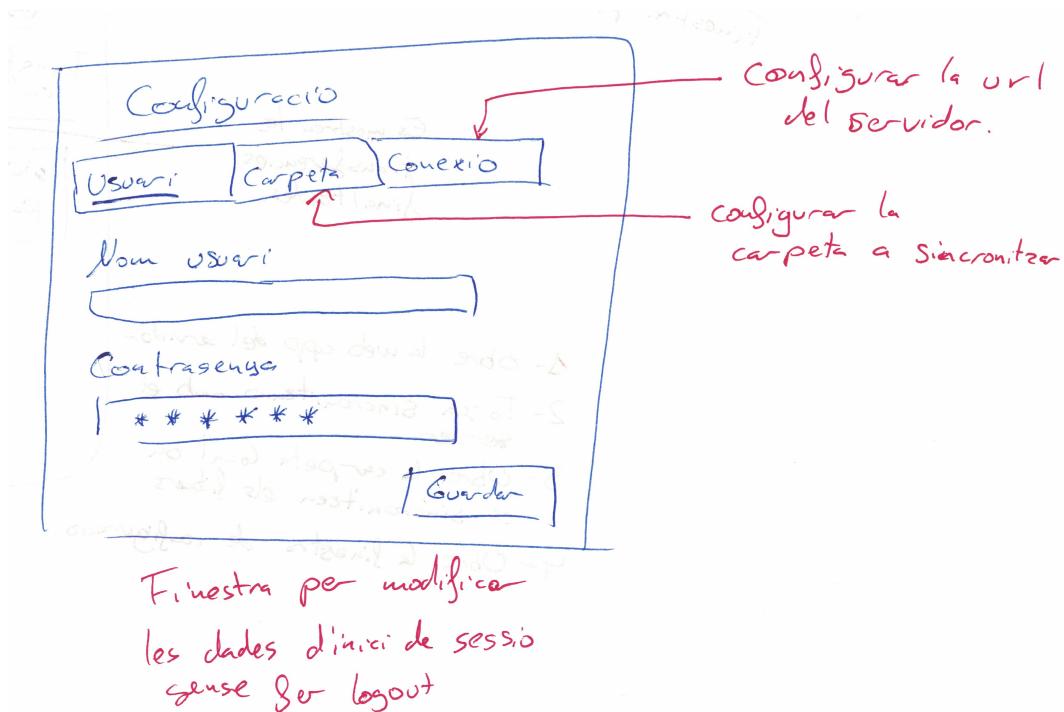


FIGURA 8.42: Menú de configuració del client d'escriptori.

Finalment, l'aplicació s'integra a la safata del sistema operatiu mitjançant una icona (Figura 8.43), des de la qual es pot accedir ràpidament a la configuració, obrir la carpeta local, veure l'estat de la sincronització o tancar l'aplicació.

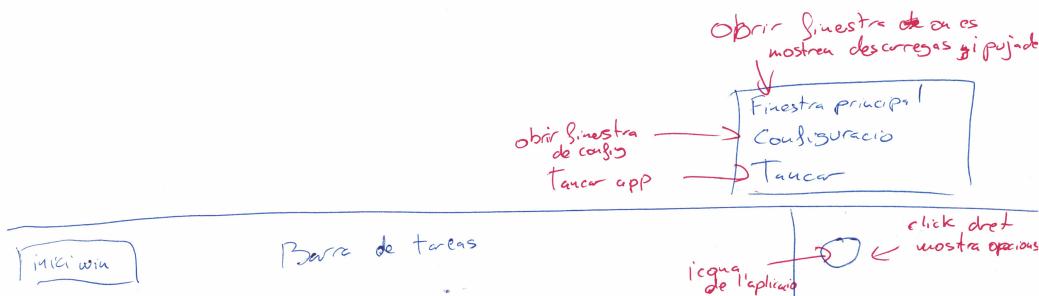


FIGURA 8.43: Menú contextual de la icona de Tauri a la safata del sistema.

8.5 Consideracions de seguretat

La seguretat ha estat un pilar fonamental en el disseny del sistema, abordant-se des de l'autenticació i autorització fins a la protecció de dades en trànsit i en repòs. A continuació, es detallen les principals mesures implementades.

8.5.1 Control d'accés basat en rols (RBAC)

El sistema implementa un model de control d'accés jeràrquic amb tres nivells de rols, gestionats pel servei UserAuthentication. Aquests rols defineixen les capacitats de cada usuari dins de l'aplicació, seguint el principi de mínim privilegi. A la Taula 8.2 es resumeixen els permisos associats a cada rol.

TAULA 8.2: Matriu de permisos per rol d'usuari.

Acció	Usuari (USER)	Administrador (ADMIN)	Superadministrador (SUPER.ADMIN)
Gestió del propi compte			
Actualitzar perfil i canviar contrasenya	✓	✓	✓
Eliminar el propi compte	✓	✓	✓
Gestió d'arxius			
Crear, llegir, actualitzar i esborrar (CRUD) arxius/carpetes propis	✓	✓	✓
Compartir arxius/carpetes amb altres usuaris	✓	✓	✓
Gestionar la paperera (restaurar/eliminar permanentment)	✓	✓	✓
Administració d'usuaris			
Llistar tots els usuaris		✓	✓
Actualitzar perfil d'usuaris amb rol USER		✓	✓
Eliminar usuaris amb rol USER		✓	✓
Administració avançada			
Actualitzar perfil d'usuaris amb rol ADMIN			✓
Eliminar usuaris amb rol ADMIN			✓
Canviar la contrasenya de qualsevol usuari			✓
Modificar el rol de qualsevol usuari			✓

A més d'aquests rols globals, el servei FileAccessControl gestiona permisos a nivell d'element individual (READ, WRITE, ADMIN), la qual cosa permet un control granular sobre qui pot accedir o modificar cada arxiu i carpeta.

8.5.2 Mecanismes de protecció

Per garantir la integritat i confidencialitat del sistema, s'han implementat diverses capes de seguretat:

- **Autenticació amb JSON Web Tokens (JWT):** Totes les peticions a l'API protegida han d'incloure un token JWT. Aquest és generat pel servei UserAuthentication utilitzant un sistema de criptografia asimètrica (RSA), signant cada token amb una clau privada. El **Gateway**, posseïdor de la clau pública, actua com a primer filtre, validant la signatura i la caducitat del token abans de redirigir la petició. Aquest mètode garanteix que només el servei d'autenticació pot generar tokens vàlids. A més, el payload del token s'enriqueix amb el rol de l'usuari i un identificador de connexió únic (connectionId). Un cop validat el token, la combinació de l'identificador d'usuari (userId) i aquesta connectionId permet identificar de manera inequívoca cada sessió d'usuari, possibilitant una autorització primerenca i un control més granular.
- **Abstracció d'identificadors interns:** El sistema ha estat dissenyat per no exposar mai els identificadors interns de la base de dades (claus primàries, generalment UUIDs) a l'exterior. En la comunicació amb els clients, s'utilitzen identificadors públics com el nom d'usuari per a referenciar usuaris o un 'elementId' específic per a fitxers i carpetes. Aquesta capa d'abstracció impedeix que un atacant que pugui interceptar la comunicació obtingui informació sobre l'estructura interna del sistema o pugui endevinar fàcilment els identificadors per accedir a recursos aliens (atac de tipus *Insecure Direct Object Reference* o IDOR).

- **Validació de dades d'entrada:** Els serveis del backend aplicuen validacions estrictes sobre les dades rebudes (p. ex., format de correu electrònic, complexitat de la contrasenya, tipus de dades esperats). Aquesta mesura és crucial per prevenir atacs d'injecció (com SQLi o XSS) i garantir la integritat de les dades.
- **Limitació de recursos:** S'estableix un límit en la mida màxima dels arxius que es poden pujar (100 MB), configurat a nivell del servei FileManagement. Aquesta mesura evita que un atacant pugui esgotar l'espai d'emmagatzematge del servidor. Tot i que no s'ha arribat a implementar un límit en la quantitat de peticions per segon (*rate limiting*), es considera una millora de seguretat crucial a desenvolupar en el futur per protegir el sistema contra atacs de denegació de servei (DoS) i de força bruta.

8.5.3 Justificació del compliment del RGPD

El disseny del sistema ha tingut en compte els requisits del Reglament General de Protecció de Dades (RGPD) des de la seva concepció:

- **Dret a l'oblit:** El sistema garanteix el dret a l'eliminació completa de les dades d'un usuari. Quan un usuari elimina el seu compte (o un administrador ho fa en nom seu), s'inicia una saga asíncrona que propaga l'ordre d'eliminació a tots els microserveis. Aquest procés assegura que totes les dades personals, arxius, permisos i registres associats siguin purgats de forma permanent de totes les bases de dades.
- **Seguretat i confidencialitat de les dades:** Les dades personals es tracten amb mesures de seguretat robustes. Les contrasenyes s'emmagatzemem a la base de dades utilitzant un algorisme de *hashing* fort, la qual cosa impedeix que puguin ser llegides fins i tot en cas d'un accés no autoritzat a la base de dades. L'accés a les dades en repòs està restringit per les credencials de cada microservei. A més, el sistema implementa una ofuscació per disseny per als arxius emmagatzemats: cada fitxer es desa al sistema d'arxius amb un identificador únic (UUID) com a nom, sense la seva extensió original, i tots junts en una única ubicació. Aquesta tècnica desacobla el contingut de les seves metadades (nom, propietari), fent que, en cas d'un accés no autoritzat al servidor, sigui extremadament difícil identificar, interpretar o associar els arxius amb usuaris concrets.
- **Príncipi de minimització de dades i accés granular:** El sistema està dissenyat per limitar l'accés a les dades només al personal autoritzat a través del sistema de rols. A més, la compartició d'arxius requereix una acció explícita per part del propietari, que pot assignar permisos de només lectura o d'escriptura, garantint que els usuaris només tinguin accés a la informació estrictament necessària.

8.6 Cobertura dels requisits no funcionals

El disseny del sistema, detallat al llarg d'aquest capítol, s'ha concebut per donar resposta directa als requisits no funcionals establerts al Capítol 6. A continuació, s'analitza com les decisions arquitectòniques i de disseny cobreixen cada una de les àrees clau.

- **Rendiment i Escalabilitat:** La decisió fonamental per satisfer aquests requisits és l'adopció d'una **arquitectura de microserveis**. Com s'ha justificat prèviament, aquesta permet l'**escalabilitat horitzontal** independent de cada servei, de manera que components amb alta demanda com FileManagement o SyncService es poden replicar per gestionar un major nombre d'usuaris i operacions simultànies. Per garantir temps de resposta ràpids, el disseny de la base de dades de cada microservei inclou **índexs estratègics** en les consultes més freqüents, com la cerca d'usuaris o la llista de fitxers d'una carpeta.
- **Seguretat:** Aquesta àrea es cobreix àmpliament a la secció de "Consideracions de seguretat". El disseny compleix els requisits d'alta prioritat mitjançant:
 - Una **autenticació segura** amb tokens JWT signats amb criptografia asimètrica (RSA).
 - Un **control d'accés granular** a dos nivells: un control global basat en rols (RBAC) i un de més fi a nivell de fitxer i carpeta gestionat pel servei FileAccessControl.
 - La **protecció contra atacs comuns** com IDOR, mitjançant l'abstracció d'identificadors interns, i la validació estricta de totes les dades d'entrada per prevenir injeccions de codi.
- **Mantenibilitat:** L'arquitectura de microserveis promou per si mateixa un codi modular i desacoblat. A més, tal com s'ha detallat a la secció de "Patrons de disseny i principis arquitectònics", cada servei segueix una **arquitectura per capes** (controlador, servei, repositori) que aplica el Principi de Responsabilitat Única (SRP). L'ús extensiu de la **injecció de dependències** i patrons com el **Repository** o el **DTO** contribueix a un codi més net, extensible i fàcil de provar.
- **Portabilitat:** L'arquitectura dissenyada està pensada per a un desplegament amb **Docker**, complint el requisit d'instal·lació senzilla a través de contenedors. La comunicació entre serveis a través d'un registre com Eureka i una passarel·la API facilita l'orquestració en qualsevol entorn compatible.
- **Usabilitat i Compatibilitat:** El disseny contempla des del principi la creació de clients diferenciats (web i escriptori amb Tauri), la qual cosa garanteix la compatibilitat amb els principals navegadors i sistemes operatius. Els esbossos de les interfícies d'usuari defineixen una estructura de vistes lògica i funcional, orientada a una experiència d'usuari intuitiva.

8.7 Conclusions

L'arquitectura modular basada en microserveis ha permès una separació clara de responsabilitats, facilitant el desenvolupament, la integració de clients diversos i el compliment de requisits com la seguretat o la sincronització. El disseny de la interfície s'ha alineat des del principi amb la funcionalitat tècnica, asssegurant una experiència d'usuari fluida.

Capítol 9

Implementació i proves

9.1 Visió general de la implementació

Aquest capítol descriu com els conceptes teòrics, els requisits i les decisions de disseny exposades en capítols anteriors s'han materialitzat en una solució de programari funcional i robusta. L'objectiu aquí no és detallar cada línia de codi, sinó oferir una visió panoràmica de l'arquitectura final, les tecnologies emprades i els patrons de comunicació que garanteixen la cohesió del sistema. La implementació que presento és el resultat directe de l'estudi de viabilitat (Capítol 2), la metodologia de treball (Capítol 3) i, especialment, de les decisions tècniques justificades al Capítol 7, tot plegat per satisfer els requisits funcionals i no funcionals definits al Capítol 6.

L'arquitectura del sistema es fonamenta en un model de microserveis, una decisió presa per garantir la modularitat, l'escalabilitat i la mantenibilitat a llarg termini, tal com es va raonar al Capítol 7. El backend està format per un conjunt de serveis independents, cadascun amb una responsabilitat única, que es despleguen en contingadors Docker i s'orquesten mitjançant un fitxer 'compose.yml'. Aquesta estratègia compleix el requisit de portabilitat (RNF-7) i facilita enormement la posada en marxa de l'entorn. Els serveis principals són:

- **UserAuthentication i UserManagement:** S'encarreguen del registre, l'autenticació (mitjançant JWT) i la gestió de les dades dels usuaris.
- **FileManagement:** Constitueix el nucli de la gestió d'arxius, gestionant la pujada, la descàrrega, la creació de carpetes i l'estructura de directoris.
- **TrashService:** Implementa la funcionalitat de la paperera de reciclatge, permetent l'eliminació temporal i la restauració d'arxius.
- **FileSharing:** Gestiona la lògica per compartir arxius entre usuaris, incloent-hi el control de permisos.
- **SyncService:** Orquestra la sincronització en temps real entre clients mitjançant WebSockets.

Aquests serveis es recolzen en components d'infraestructura com **Spring Cloud Gateway**, que actua com a única porta d'entrada per a totes les peticions externes, centralitzant la seguretat i l'enrutament, i **Eureka**, que proporciona el descobriment de serveis per a una comunicació interna resistent. Al front, tenim un **client web** desenvolupat amb React i un **client d'escriptori** natiu construït amb Tauri i Svelte, ambdós

dissenyats per oferir una experiència d'usuari moderna i eficient.

La comunicació entre aquests components segueix dos patrons principals, una decisió clau explicada al Capítol 7. Per a les operacions que requereixen una resposta immediata per part de l'usuari (com iniciar sessió o pujar un arxiu), s'utilitza una comunicació síncrona mitjançant **peticions HTTP**. Aquestes peticions flueixen des del client, a través del Gateway, fins al microservei corresponent. En canvi, per a processos que poden executar-se en segon pla sense bloquejar l'usuari (com la neteja de dades en cascada després d'eliminar un compte), he implementat un sistema de **missatgeria asíncrona amb RabbitMQ**. Aquest enfocament millora la resiliència i l'experiència d'usuari, ja que garanteix que les tasques es compleixin de manera fiable fins i tot si un servei pateix una fallada temporal. Finalment, la sincronització en temps real es duu a terme a través d'una connexió **WebSocket** persistent, que permet al servidor notificar canvis a tots els clients connectats de manera instantània.

Els fluxos complets de cada funcionalitat, que demostren la interacció entre tots aquests components per satisfer els requisits del sistema, es troben detallats a l'Apèndix A (@AppendixA.tex, casos d'ús UC-01 a UC-22).

En lloc de presentar fragments de codi aïllats, considero més il·lustratiu referir-se a la taula de versions i dependències principals que ja vaig detallar a la Taula 5.1 del Capítol 5. Aquesta taula proporciona una fotografia precisa de la pila tecnològica final del projecte, on destaquen **Java 17 amb Spring Boot 3** per al backend, **React 18** per al client web i **Tauri 1.5 amb Rust** per al client d'escriptori. Aquestes eleccions són el resultat de l'anàlisi i les decisions preses al llarg de tot el desenvolupament.

En resum, la implementació del projecte tradueix una arquitectura distribuïda a un sistema tangible. El flux global es basa en una combinació estratègica de comunicació HTTP per a la interacció directa i missatgeria asíncrona per a la resiliència i el desacoblamet, demostrant com l'aplicació pràctica de patrons de disseny moderns permet construir solucions complexes, escalables i robustes.

9.2 Entorn de desenvolupament i desplegament

Per garantir la coherència i la reproduïibilitat del sistema, un dels requisits clau del projecte (RNF-4, RNF-7), vaig prestar una atenció especial a la definició de l'entorn de desenvolupament i al procés de desplegament. La decisió fonamental va ser estructurar tot el projecte en un **monorepo**, una elecció estratègica per simplificar la gestió de dependències i assegurar que totes les peces del sistema (backend, frontend, client d'escriptori i documentació) evolucionessin de manera sincronitzada. Aquesta centralització facilita la traçabilitat dels canvis i redueix la complexitat inherent a la gestió de múltiples repositoris.

El cor de l'entorn de desenvolupament local és el fitxer docker-compose.yml, que actua com a orquestrador de tota l'arquitectura de microserveis. Com es va justificar al Capítol 7, l'ús de Docker i Docker Compose va ser una decisió presa per la seva simplicitat i per l'experiència prèvia, permetent aixecar un ecosistema complex amb una única comanda: docker compose up. Aquest fitxer no només defineix cada microservei, sinó també les seves relacions, les xarxes, els volums persistents i la configuració essencial.

¹ file - manager :

```

2   build: ./FileManagement
3   container.name: filemanagement
4   environment:
5     - "SPRING_PROFILES_ACTIVE=docker"
6   depends_on:
7     eureka:
8       condition: service.healthy
9   volumes:
10    - ./storage/data:/app/files

```

LISTING 9.1: Fragment del fitxer ‘compose.yml’ definint un servei

Com es pot observar en aquest fragment, cada servei es construeix a partir del seu propi directori (p. ex., ./FileManagement), se li assigna un perfil de Spring específic per a Docker (SPRING_PROFILES_ACTIVE=docker) i s'estableixen dependències explícites. En aquest cas, el servei file-manager no s'iniciarà fins que Eureka estigui saludable (service.healthy), garantint un ordre d'arrencada correcte i evitant errors en cascada. A més, es munta un volum local (./storage_data) per persistir els arxius pujats pels usuaris, una configuració crítica per al desenvolupament i les proves.

Per simplificar encara més la posada en marxa, especialment per a usuaris amb menys experiència tècnica, vaig desenvolupar un script d'instal·lació (setup.sh), tal com es va mencionar al Capítol 5. Aquest script automatitza tots els passos necessaris: comprova les dependències del sistema (com Docker i Java), construeix les imatges de tots els contenidors i finalment invoca docker compose up per llançar la plataforma. Això redueix el procés d'instal·lació a l'execució d'un sol fitxer, complint l'objectiu de fer el sistema accessible.

La configuració dels serveis es gestiona principalment a través de **variables d'entorn** i els fitxers de propietats de Spring Boot, que inclouen perfils per a diferents entorns (com dev per a desenvolupament local i docker per a l'execució dins de contenidors). Les variables d'entorn crítiques, com les credencials de la base de dades o les claus per a la signatura de tokens JWT, es defineixen directament al compose.yml o en fitxers .env per evitar exposar informació sensible al codi font, una pràctica de seguretat fonamental.

9.3 Implementació del backend

A continuació, s'analitza la implementació de cada bloc funcional del backend, posant èmfasi en com els microserveis materialitzen els requisits del sistema i les decisions d'arquitectura preses.

9.3.1 Gestió d'usuaris (UserAuthentication)

El microservei UserAuthentication és el pilar de la seguretat del sistema. La seva responsabilitat exclusiva és gestionar el cicle de vida de l'autenticació dels usuaris, una decisió de disseny (Capítol 7) que permet centralitzar i aïllar la lògica de seguretat més crítica. Internament, segueix una arquitectura per capes clàssica de Spring Boot:

- **Capa de controladors (controllers):** Exposa els endpoints REST que el client consumeix per a les operacions d'autenticació.

- **Capa de serveis (services):** Conté la lògica de negoci principal, com la validació de credencials, el registre de nous usuaris i la gestió de tokens.
- **Capa de repositoris (repositories):** Abstrau l'accés a la base de dades mitjançant Spring Data JPA.

Aquest servei és el principal responsable de la implementació dels següents casos d'ús, els detalls dels quals es poden trobar a l'Apèndix A ([Casos d'Ús](#)) i els diagrames d'activitat a l'Apèndix B ([Diagrams d'Activitat](#)):

- **UC-01: Registrar-se:** El controlador rep la petició de registre i delega al servei la tasca d'orquestrar la creació de l'usuari. Aquest procés inclou la validació de les dades, la comunicació amb altres serveis com UserManagement i FileManagement, i la generació de l'estat inicial de sincronització, tal com es detalla al diagrama d'activitat [UC-01](#).
- **UC-02: Iniciar sessió:** S'encarrega de validar les credencials i, si són correctes, generar els tokens JWT d'accés i de refresh.
- **UC-15: Canviar contrasenya:** Verifica la contrasenya antiga de l'usuari abans de xifrar i emmagatzemar la nova.
- **UC-16: Eliminar compte:** Inicia el procés de purga de dades de l'usuari de forma asíncrona, enviant un missatge a RabbitMQ que serà consumit per la resta de serveis.

L'eix central de l'autenticació és la generació i validació de tokens JWT, una responsabilitat encapsulada a la classe JwtTokenUtil. Aquesta utilitat, que es basa en la llibreria io.jsonwebtoken, utilitza un parell de claus RSA (una privada per signar i una pública que el Gateway podria fer servir per verificar) per garantir la integritat i autenticitat dels tokens, una mesura de seguretat fonamental.

```

1 public String generateToken(UserEntity user, boolean isRefreshToken) ->
2     long validity = isRefreshToken ? REFRESHTOKENVALIDITY :
3         ACCESSTOKENVALIDITY;
4     Map<String, Object> claims = new HashMap<>();
5     claims.put("role", user.getRole());
6     claims.put("connection-id", UUID.randomUUID().toString());
7     return createToken(claims, user.getUsername(), validity);
8
9     private String createToken(Map<String, Object> claims, String username,
10         long validity) ->
11         return Jwts.builder()
12             .setClaims(claims)
13             .setSubject(username)
14             .setIssuedAt(new Date(System.currentTimeMillis()))
15             .setExpiration(new Date(System.currentTimeMillis() +
validity))
16             .signWith(privateKey).compact();

```

LISTING 9.2: Generació de tokens a la classe 'JwtTokenUtil'

Com es pot observar, cada token conté no només la identitat de l'usuari (subject) i la seva data d'expiració, sinó també *claims* addicionals com el seu role, que serà crucial per a la lògica d'autorització al Gateway i altres serveis. La gestió de secrets, com la

clau privada, es realitza carregant-la des del *classpath* a l'inici de l'aplicació, evitant així la seva exposició directa al codi.

El controlador principal, AuthenticationController, exposa els endpoints REST que permeten la interacció amb el client.

```

1  @PostMapping("/register")
2  public ResponseEntity<?> registerUser(@RequestBody @Valid
3      UserRegisterRequest userInfoRequest, HttpServletResponse response) -
4      // ... validació de l'usuari i email ...
5      try {
6          userService.register(userInfoRequest);
7          UserEntity user = userService.loadUserByUsername(userInfoRequest
8              .getUsername());
9          return addTokenHeaders(user);
10         " catch (IllegalArgumentException e) -
11             return ResponseEntity.badRequest().body(e.getMessage());
12         "
13     @PostMapping("/login")
14     public ResponseEntity<?> loginUser(@RequestBody UserInfoRequest
15         userInfoRequest) -
16         if (!userService.checkCredentials(userInfoRequest.getUsername(),
17             userInfoRequest.getPassword())) -
18             return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Invalid credentials");
19         "
20         UserEntity user = userService.loadUserByUsername(userInfoRequest
21             .getUsername());
22         return addTokenHeaders(user);
23     "

```

LISTING 9.3: Endpoints de registre i login a 'AuthenticationController'

En el procés de login, si les credencials són correctes, s'invoca el mètode `addTokenHeaders`, que genera un token d'accés de curta durada (15 minuts) i un de refresh de llarga durada (7 dies). Aquests s'envien al client a les capçaleres de la resposta, una pràctica de seguretat que facilita la seva gestió al frontend. La gestió d'errors es realitza retornant codis d'estat HTTP apropiats (com 400 Bad Request o 401 Unauthorized) i missatges clars, un aspecte clau per a la depuració i una bona experiència de desenvolupador.

9.3.2 Gateway i filtre de JWT

El microservei Gateway, basat en Spring Cloud Gateway, és la porta d'entrada única per a totes les peticions externes, una peça clau en l'arquitectura de microserveis que vaig escollir, tal com vaig justificar al Capítol 7. La seva funció no es limita a ser un simple proxy; actua com un vigilant que centralitza la seguretat i organitza el trànsit cap als diferents serveis interns. Totes les rutes de l'aplicació estan definides de manera declarativa en una classe de configuració, `GatewayConfig.java`, mitjançant un bean de `RouteLocator`.

```

1 .route("user/login/post", r -> r.path("/users/auth/login"))
2     .and() .method(HttpMethod.POST, HttpMethod.OPTIONS)
3     .uri("lb://USERAUTHENTICATION"))
4
5 .route("files/root/get", r -> r.path("/files/root"))

```

```

6     . and() . method (HttpMethod.GET, HttpMethod.OPTIONS)
7     . filters (f -> f . filter (jwtAuthenticationFilter))
8     . uri ("lb://FileManagement"))
9
10    . route ("admin'users'get" , r -> r . path ("/admin/users"))
11    . and() . method (HttpMethod.GET, HttpMethod.OPTIONS)
12    . filters (f -> f . filter (jwtAuthenticationFilter))
13    . uri ("lb://UserManagement"))

```

LISTING 9.4: Exemples de definició de rutes a 'GatewayConfig'

Aquest enfocament permet una gestió centralitzada i clara de l'enrutament. Per exemple:

- La ruta user_login_post (**UC-02**) és pública; les peticions a /users/auth/login es redirigeixen directament al servei USERAUTHENTICATION sense cap filtre de seguretat.
- La ruta files_root_get, en canvi, aplica el filtre jwtAuthenticationFilter abans de redirigir la petició al servei FileManagement. Això assegura que només els usuaris autenticats puguin accedir als seus arxius.
- La ruta admin_users_get (**UC-05**) també passa pel filtre, que, com veurem, conté lògica específica per validar rols administratius.

El component més crític del Gateway és el filtre global JwtAuthenticationFilter. Aquest filtre intercepta cada petició (excepte les públiques), extreu el token JWT de la capçalera Authorization i orquestra la seva validació.

```

1 @Component
2 public class JwtAuthenticationFilter implements GatewayFilter -
3   // ... injecció de WebClient i altres dependències ...
4
5   @Override
6   public Mono filter(ServerWebExchange exchange ,
7     GatewayFilterChain chain) -
8     ServerHttpRequest request = exchange.getRequest ();
9     // ... extracció del token de la capçalera ...
10
11   String accessToken = authHeader.substring (7);
12
13   return webClient.method (HttpMethod.POST)
14     . uri ("/users/auth/check") // Crida interna a
15     UserAuthentication
16     . header (HttpHeaders.AUTHORIZATION, "Bearer " +
17     accessToken)
18     . retrieve ()
19     . onStatus (HttpStatusCode::isError , response -> Mono.
20     error (new InvalidTokenException ()))
21     . bodyToMono (UserAuthResponse . class )
22     . flatMap (userAuthResponse -> -
23       // Validació de rols per a rutes /admin/
24       if (request.getPath () . toString () . startsWith ("/admin/
")) &&
25         !( "ADMIN" . equals (userAuthResponse . getRole ()) || "
26         SUPERADMIN" . equals (userAuthResponse . getRole ())) -
27           exchange . getResponse () . setStatusCode (HttpStatus .
28           FORBIDDEN) ;
29           return exchange . getResponse () . setComplete ();
30

```

```

25          // Enriquiment de la petició amb dades de l'usuari
26          ServerHttpRequest modifiedRequest = request.mutate()
27                  .header("X-User-Id", userAuthResponse.getId
28                      () .toString())
29                  .header("X-Connection-Id", userAuthResponse.
30                      getConnectionId())
31                  .build();
32          return chain.filter(exchange.mutate().request(
33              modifiedRequest).build());
34      }
35      .onErrorResume(e -> {
36          exchange.getResponse().setStatusCode(HttpStatus.
37              UNAUTHORIZED);
38          return exchange.getResponse().setComplete();
39      });
40  
```

LISTING 9.5: Implementació del filtre 'JwtAuthenticationFilter'

En lloc de contenir la lògica de validació del token directament, el filtre delega aquesta responsabilitat al servei UserAuthentication fent una crida interna al seu endpoint /users/auth/check. Aquesta decisió de disseny manté el Gateway lleuger i respecta el principi de responsabilitat única. Si la resposta del servei d'autenticació és positiva, el filtre realitza dues accions clau:

- Validació de rols:** Comprova si la ruta requereix permisos d'administrador i si l'usuari els té. Si no, retorna un error **403 Forbidden**, indicant que l'usuari està autenticat però no autoritzat per a aquest recurs. Això és diferent d'un **401 Unauthorized**, que es retorna quan el token és invàlid o no es proporciona.
- Enriquiment de la petició:** Afegeix les capçaleres X-User-Id i X-Connection-Id a la petició abans de passar-la al microservei de destinació. Això permet que els serveis interns confiïn en aquestes capçaleres per identificar l'usuari sense necessitat de validar el token de nou, optimitzant el rendiment.

Tot i que aquest disseny és robust, soc conscient que la crida interna des del Gateway a UserAuthentication a cada petició protegida introduceix una latència addicional i podria convertir-se en un coll d'ampolla sota càrregues elevades. Una possible millora futura, discutida al Capítol 12, seria implementar un mecanisme de memòria cau (cache) al Gateway per als resultats de validació de tokens o compartir la clau pública de signatura perquè el Gateway pugui validar els tokens localment sense la crida de xarxa.

9.3.3 Gestió d'arxius (FileManagement)

El microservei FileManagement és el cor operacional del sistema, responsable de tota la lògica relacionada amb la creació, modificació, consulta i eliminació d'arxius i carpetes. La seva implementació materialitza un gran nombre de casos d'ús, com **UC-08** (Crear/Pujar), **UC-09** (Renomenar, Moure, Copiar) i **UC-10** (Descarregar), els detalls dels quals es poden consultar als apèndixs A i B.

L'arquitectura d'aquest servei, igual que la dels altres, segueix el patró per capes de Spring Boot. El controlador principal, ElementController, exposa una API REST

completa per a la manipulació d'elements. Aquest delega la lògica de negoci a diferents serveis especialitzats: ElementService per a operacions genèriques, i FileService i FolderService per a la lògica específica de fitxers i carpetes, respectivament.

Una de les responsabilitats clau del controlador és rebre les peticions i orquestrar les accions. Per exemple, en la creació d'un element (**UC-08**), l'endpoint POST /files gestiona tant la creació de carpetes com la pujada de fitxers.

```

1 @PostMapping( produces = MediaType.APPLICATION_JSON_VALUE)
2 public ResponseEntity<?> createElement(
3     @RequestHeader("X-User-Id") UUID userId ,
4     @RequestHeader("X-Connection-Id") UUID connectionId ,
5     @RequestPart("request") CreateFileRequest request ,
6     @RequestPart(value = "file", required = false) MultipartFile
    file
7 ) throws IOException -
8     FolderEntity parent = folderService.getFolderById(request.
getparentId(), false);
9     fileAccessService.checkAccessFolder(parent, userId, AccessType.WRITE
());
10
11    if (request.isFolder()) -
12        FolderEntity newFolder = folderService.createFolder(request.
getName(), parent, userId, false, connectionId.toString());
13        // ... notificació a SyncService ...
14        return new ResponseEntity<?>(folderMapper.map(newFolder),
15 HttpStatus.CREATED);
16    " else -
17        FileEntity newFile = fileService.createFile(file, parent, userId
, connectionId.toString());
18        // ... notificació a SyncService ...
19        return new ResponseEntity<?>(fileMapper.map(newFile), HttpStatus.
CREATED);
20    "

```

LISTING 9.6: Endpoint per a la creació d'elements a 'ElementController'

En aquest fragment es pot observar com, abans de qualsevol operació, es realitza una crida a un altre microservei (FileAccessService) per verificar els permisos de l'usuari, una decisió de disseny per desacoblar la lògica de negoci de la de control d'accés.

La persistència de les dades es gestiona a través de repositoris JPA. En aquest servei, he modelat una jerarquia d'entitats on FileEntity i FolderEntity hereten d'una classe base ElementEntity, permetent així tractar-los de manera polimòrfica en moltes operacions. El repositori ElementRepository s'encarrega de les operacions comunes.

```

1 public interface ElementRepository extends JpaRepository<ElementEntity,
2     UUID> -
3     // Spring Data JPA genera automàticament les implementacions

```

LISTING 9.7: Repositori per a l'entitat 'ElementEntity'

Un dels reptes tècnics més interessants en la gestió d'arxius és el maneig de les operacions sobre estructures de directoris, especialment el moviment de carpetes (**UC-09B**). Per evitar estats inconsistents, com ara moure una carpeta dins d'ella mateixa o d'una de les seves subcarpetes, vaig implementar una validació de dependències

circulars. Aquesta lògica es troba al FolderService i s'executa dins d'una transacció per garantir l'atomicitat de l'operació.

```

1  @Transactional
2  public FolderEntity updateFolderMetadata(UUID folderId, String name,
3      UUID parentId) {
4      FolderEntity folder = folderRepository.findById(new
5          ElementEntity(folderId))
6          .orElseThrow(() -> new ResponseStatusException(HttpStatus.
7              NOTFOUND, "Folder not found"));
8
9      if (parentId != null && !parentId.equals(folder.getParent() .
10          getElementId())) {
11          FolderEntity newParent = folderRepository.findById(new
12              ElementEntity(parentId))
13              .orElseThrow(() -> new ResponseStatusException(
14                  HttpStatus.NOTFOUND, "Parent folder not found"));
15
16          // Mètode clau per a la validació
17          validateNoCircularDependency(folder, newParent);
18
19          folder.setParent(newParent);
20      }
21
22      // ... actualització del nom ...
23      return folderRepository.save(folder);
24
25
26
27 }
```

LISTING 9.8: Validació de dependències circulars a 'FolderService'

El mètode validateNoCircularDependency recorre l'arbre de directoris cap amunt des de la carpeta de destinació (targetParent). Si en aquest camí troba la carpeta que s'està intentant moure (elementToMove), significa que es produiria una dependència circular, de manera que lanza una excepció personalitzada que resulta en un error 400 Bad Request per al client. L'ús de l'anotació @Transactional assegura que si aquesta validació (o qualsevol altra part de l'operació) falla, tots els canvis a la base de dades es desfan, mantenint la integritat de les metadades dels arxius.

9.3.4 Compartició (FileSharing)

El microservei FileSharing gestiona la complexa lògica de compartir arxius i carpetes entre usuaris, una de les funcionalitats centrals del sistema. La seva implementació està directament lligada als casos d'ús **UC-13** (Compartir), **UC-13A** (Revocar accés) i **UC-13B** (Deixar de seguir un element compartit). Com es descriu al Capítol 7, la comunicació asíncrona juga aquí un paper crucial per mantenir la coherència entre els diferents clients.

Quan un usuari comparteix un element (**UC-13**), el controlador del servei FileSharing invoca el mètode shareElement del FileSharingService. Aquest mètode orquestra una sèrie d'accions fonamentals per garantir la seguretat i la integritat del procés:

1. **Validació de permisos:** Primer, realitza una crida síncrona mitjançant Feign al microservei FileAccessControl per comprovar que l'usuari que fa la petició té permisos d'ADMIN sobre l'element. Aquesta és una mesura de seguretat crítica per evitar que usuaris no autoritzats puguin compartir contingut.
2. **Creació de regles d'accés:** Si la validació és correcta, es crea la regla d'accés corresponent per a l'usuari destinatari, de nou mitjançant una crida a FileAccessControl.
3. **Propagació de permisos:** El servei propaga de manera recursiva els permisos a tots els arxius i subcarpetes continguts dins de l'element compartit.
4. **Notificació asíncrona:** Un cop completada l'operació, el servei publica un esdeveniment a una cua de RabbitMQ. Aquest missatge conté els detalls de l'acció realitzada.

```

1  @Transactional
2  public void shareElement(UUID userId, ShareRequest shareRequest, String
3      connectionId) {
4      // 1. Validació de permisos (crida a FileAccessControl)
5      AccessResponse requesterAccess = fileAccessControlClient.
6          getFileAccess(
7              shareRequest.getElementId(), userId);
8      if (requesterAccess == null || requesterAccess.getAccessType() !=
9          AccessType.ADMIN.ordinal()) {
10         throw new ResponseStatusException(HttpStatus.FORBIDDEN, "Admin
11             permission required");
12     }
13
14     // 2. Obtenció de l'usuari destinatari (crida a UserManagement)
15     UUID shareWithUserId = userManagementClient.getUserId(shareRequest.
16         getUser());
17
18     // 3. Creació i propagació de regles d'accés
19     AccessRequest accessRequest = new AccessRequest(shareWithUserId,
20         shareRequest.getElementId(), shareRequest.getAccessType().ordinal());
21     fileAccessControlClient.createAccess(accessRequest);
22
23     // ... lògica per actualitzar fills i marcar l'element com a
24     //     compartit ...
25
26     // 4. Publicació de l'esdeveniment a RabbitMQ
27     commandService.sendShared(shareRequest.getElementId(), connectionId,
28         userId.toString(), parentIdString);
29 }
```

LISTING 9.9: Mètode principal del servei 'FileSharingService'

La revocació d'accés (**UC-13A** i **UC-13B**) segueix un patró similar: el servei valida els permisos i elimina les regles d'accés corresponents, notificant posteriorment el canvi a través de RabbitMQ amb una acció de tipus unshared-element.

A l'altre extrem de la comunicació, el microservei SyncService disposa d'un component "listener" que escolta constantment la cua de RabbitMQ. Quan rep un missatge de compartició, el processa i notifica als clients connectats a través de WebSocket.

```

1  @Component
2  public class Receiver -
3      // ... injecció de dependències ...
4
5  @RabbitListener(queues = RabbitConfig.COMMANDQUEUE)
6  public void processCommand(Map<String, Object> payload) -
7      CommandRabbit command = new CommandRabbit(* ... mapeig del
8      payload ... *);
9
9      // Si l'acció és de compartir/deixar de compartir
10     if(command.action().equals("shared-element") — command.action()
11         .equals("unshared-element")) -
12         // Notifica directament als clients via WebSocket
13         websocketService.sendWebCommand(command.action(), command.
14         userId(), command.connectionId(), command.elementId(), command.
15         parentId());
16         " else -
17         // Per a altres accions (CRUD), actualitza el snapshot i l'
18         envia
19         SnapshotEntity snapshot = snapshotService.processCommand(
20             command);
21         websocketService.sendSnapshot(snapshot, command.userId(),
22             command.connectionId());
23         "
24         "
25     "
26 "
27 "
28 "
29 "

```

LISTING 9.10: Listener de RabbitMQ a 'SyncService'

Aquest disseny desacoblat és un dels punts clau de l'arquitectura. El servei FileSharing no necessita saber quins clients estan connectats ni com notificar-los. La seva única responsabilitat és publicar un fet ("s'ha compartit un element") al sistema de missatgeria. És el SyncService qui s'encarrega de la lògica de difusió. Aquest patró millora l'escalabilitat i la resiliència. Si el SyncService estigués temporalment caigut, els missatges de compartició s'acumularien a la cua de RabbitMQ i es processarien un cop el servei es recuperés, garantint la consistència final del sistema sense perdre cap esdeveniment. A més, en el processament d'aquests missatges, és fonamental garantir la **idempotència**: si un missatge es processa més d'una vegada per error, el resultat final ha de ser el mateix que si s'hagués processat una sola vegada, evitant així inconsistències.

9.3.5 Papelera i eliminació asíncrona (TrashService)

El microservei TrashService implementa la funcionalitat de la paperera de reciclatge i, més important encara, orquestra el complex procés d'eliminació permanent de dades, una operació crítica per complir amb el "dret a l'oblit" estipulat pel RGPD. La seva implementació cobreix els casos d'ús **UC-11** (Enviar a la paperera), **UC-12** (Eliminar permanent), **UC-17** (Restaurar) i juga un paper central en l'**UC-07** (Eliminació d'usuari).

Quan un usuari envia un element a la paperera (**UC-11**), el sistema no realitza una eliminació física immediata. En lloc d'això, el TrashService executa un borrat lògic.

```

1  public void moveToTrash(UUID userId, UUID connectionId, UUID elementId)
2      -
3          // 1. Validació de permisos d'escriptura sobre l'element
4          // ... crida a FileAccessControlClient ...

```

```
5 // 2. Crida a FileManagement per marcar l'element i els seus fills  
6 com a "eliminats"  
7 SetDeletedRequest request = new SetDeletedRequest(true);  
8 SetDeletedResponse response = fileManagementClient.  
9 setElementDeletedState(userId, connectionId, elementId, request);  
10  
11 // 3. Creació o actualització d'un TrashRecord per a cada element  
12 // afectat  
13 List<UUID> affectedIds = response.getElementIds();  
14 for (UUID affectedId : affectedIds) -  
15     boolean isRoot = affectedId.equals(elementId);  
         updateOrCreateRecord(userId, affectedId, isRoot);  
     "  
 "
```

LISTING 9.11: Mètode per moure un element a la paperera a ‘TrashService’

Aquest procés crea un registre TrashRecord per a cada element mogut, que inclou la data d'eliminació i una data de caducitat. La restauració (**UC-17**) simplement inverteix aquest procés, marcant els elements com a actius de nou i eliminant els registres de la paperera.

L'eliminació permanent (**UC-12**) és on l'arquitectura asíncrona demostra el seu valor. Quan un usuari buida la paperera o un element caduca, el TrashService no intenta eliminar totes les dades de cop. En canvi, inicia un procés de purga en segon pla:

1. L'element físic s'elimina del sistema d'arxius a través de FileManagement.
 2. El TrashRecord corresponent es marca amb l'estat PENDING_DELETION.
 3. Un procés programat (*scheduler*) o un listener de cua detecta aquests registres pendents. Per a cada un, envia missatges a RabbitMQ, dirigits a altres serveis com FileAccessControl i FileSharing, ordenant-los que eliminin qualsevol dada relacionada (regles d'accés, registres de compartició, etc.).

Aquest mateix patró de coreografia basat en esdeveniments s'utilitza per a l'eliminació completa d'un usuari (**UC-07**). El procés s'inicia amb un missatge a la cua `USER_DELETED_QUEUE`, que és rebut pel servei `Trash`.

```
1  @Component
2  public class Receiver {
3      @Autowired
4      private UserDeletionService userDeletionService;
5
6      // ... altres listeners ...
7
8      @RabbitListener(queues = RabbitConfig.USERDELETEDQUEUE)
9      public void receiveUserDeletedMessage(String userId) {
10         // Inicia el procés d'eliminació orquestrada de l'usuari
11         userDeletionService.startUserDeletion(UUID.fromString(userId));
12     }
13
14     @RabbitListener(queues = RabbitConfig.
15     USERDELETIONCONFIRMATIONQUEUE)
16     public void receiveDeletionConfirmation(Map<String, Object> payload) {
17         // Processa les confirmacions de cada servei que ha acabat la
18         // seva part de la neteja.
```

```

17     " userDeletionService . processDeletionConfirmation ( confirmation ) ;
18
19 "

```

LISTING 9.12: Listener de RabbitMQ per a la neteja asíncrona a 'TrashService'

El UserDeletionService s'encarrega d'enviar missatges a tots els microserveis rellevants i esperar les seves confirmacions. Aquest disseny garanteix la **tolerància a fallades**: si un servei no pot completar la seva part de la neteja immediatament, el sistema pot reintentar l'operació més tard sense perdre la petició. Aquest mecanisme de *retries* i confirmacions assegura que, eventualment, totes les dades de l'usuari s'eliminaran de forma completa i consistent, complint així amb els requisits legals i tècnics de manera robusta i resilient.

9.3.6 Servei de sincronització (SyncService)

El microservei SyncService és el component final de l'arquitectura del backend i el responsable d'una de les funcionalitats més complexes i importants: mantenir tots els clients sincronitzats en temps real (**UC-14**). La seva principal responsabilitat és gestionar les connexions WebSocket, processar els esdeveniments de canvi publicats per altres serveis i difondre les actualitzacions als clients corresponents.

L'arquitectura del servei es basa en diversos components clau:

- **Gestor de WebSockets:** Un WebSocketHandler que gestiona el cicle de vida de les connexions dels clients (web i escriptori).
- **Consumidor de RabbitMQ:** Un *listener* que se subscriu a la cua d'esdeveniments on altres microserveis (com FileManagement o FileSharing) publiquen canvis.
- **Servei de Snapshots:** Un servei que manté a la base de dades una representació de l'arbre de fitxers de cada usuari (un "snapshot") i el reconstrueix cada cop que es produeix un canvi.

El flux de connexió s'inicia quan un client vol establir una comunicació en temps real. El procés de connexió, que passa obligatòriament pel Gateway, es mostra a la configuració del SyncService.

```

1 @Configuration
2 @EnableWebSocket
3 public class WebSocketConfig implements WebSocketConfigurer -
4     @Override
5     public void registerWebSocketHandlers (@NonNull
6         WebSocketHandlerRegistry registry) -
7         RawWebSocketHandler handler = rawWebSocketHandler () ;
8
9         // Endpoint per al client d'escriptori
10        registry.addHandler (handler , "/websocket")
11            . setAllowedOrigins (" * ")
12            . addInterceptors (new UserIdHandshakeInterceptor ()) ;
13
14         // Endpoint per al client web
15        registry.addHandler (handler , "/websocket/web")
16            . setAllowedOrigins (" * ")
17            . addInterceptors (new UserIdHandshakeInterceptor ()) ;

```

```

17 "
18
19     @Bean
20     public RawWebSocketHandler rawWebSocketHandler() -
21         return new RawWebSocketHandler();
22 "
23 "

```

LISTING 9.13: Configuració del WebSocket a 'SyncService'

L'interceptor UserIdHandshakeInterceptor s'encarrega d'extreure l'ID de l'usuari i l'ID de connexió del token JWT durant la negociació inicial, injectant-los als atributs de la sessió WebSocket. Un cop la connexió arriba al RawWebSocketHandler, aquest guarda la sessió en un mapa en memòria, diferenciant entre connexions web i d'escriptori. Per a les connexions d'escriptori, envia immediatament un *snapshot* complet de l'arbre de fitxers de l'usuari.

El nucli de la sincronització rau en el processament dels esdeveniments. Quan el consumidor de RabbitMQ rep un missatge (p. ex., "s'ha creat un fitxer"), invoca el SnapshotService per actualitzar la representació de l'arbre de fitxers a la base de dades.

```

1 @Transactional
2 public SnapshotEntity processCommand(CommandRabbit command) -
3     // Troba el snapshot de l'usuari
4     SnapshotEntity snapshot = snapshotRepository.findById(UUID.
5         fromString(command.userId()));
6     // ... lògica per trobar l'element pare i l'element a modificar ...
7
8     if (command.action().equals("delete")) -
9         snapshot = deleteElement(targetElement, parent, snapshot);
10    " else -
11        if (targetElement == null) -
12            snapshot = addElement(command, parent, snapshot);
13        " else -
14            snapshot = modifyElement(command, targetElement, parent,
15            snapshot);
16
17    // Un cop modificat l'arbre, es recalcula el hash propagant-lo cap
18    // amunt
19    recalculateHash(parent);
20
21    "
22
23 private void recalculateHash(SnapshotElementEntity element) -
24     if (element != null && "folder".equals(element.getType())) -
25         String newHash = calculateFolderHash(element);
26         element.setHash(newHash);
27         snapshotElementRepository.save(element);
28
29     if (element.getParent() != null) -
30         // Crida recursiva cap a l'arrel
31         recalculateHash(element.getParent());
32
33    "

```

34

"

LISTING 9.14: Processament de comandes i reconstrucció de hash a 'SnapshotService'

Un cop l'snapshot s'ha actualitzat, el Receiver notifica al WebSocketHandler, que s'encarrega de difondre el canvi. La lògica de difusió és diferent segons el tipus de client:

- **Per al client web:** S'envia un esdeveniment simple, com `updated_tree`, que indica al client que ha de tornar a demanar l'arbre de fitxers de la carpeta actual.
- **Per al client d'escriptori:** S'envia el *snapshot* complet actualitzat. El client d'escriptori compara el nou hash de l'arrel amb el que tenia; si són diferents, substitueix el seu estat local pel nou *snapshot*.

Per garantir la robustesa, he implementat mecanismes per gestionar la pèrdua de connexions (*idle timeouts*) i una estratègia de reconnexió simple per part dels clients. El sistema es basa en la **consistència eventual**: tot i que poden existir petites latències, es garanteix que tots els clients acabaran rebent tots els canvis. La gestió de l'estat de les connexions en memòria és un punt crític; en un escenari de producció a gran escala, aquest estat s'hauria d'externalitzar a un magatzem compartit com Redis per permetre l'escalat horitzontal del propi SyncService.

Apèndix A

Fitxes completes de Casos d'Ús

A continuació es detallen els casos d'ús principals del sistema, descrivint la interacció entre els actors i els diferents microserveis.

A.1 UC-01: Registrar-se

- **Descripció:** L'usuari crea un compte nou.
- **Actors:** Usuari.
- **Precondicions:** No haver iniciat sessió.
- **Postcondicions:** Compte creat i sessió iniciada.
- **Escenari principal:**
 1. El client envia una sol·licitud de registre al servei **UserAuthentication** a través del Gateway.
 2. El servei valida les dades rebudes:
 - Comprova que el format del nom d'usuari, contrasenya i email siguin correctes.
 - Verifica que el nom d'usuari no estigui ja en ús.
 - Consulta a **UserManagement** per assegurar que l'email no estigui registrat.
 3. Si les validacions són correctes, **UserAuthentication** guarda les credencials (nom d'usuari i contrasenya xifrada) i inicia la creació de dades en altres serveis:
 - Fa una crida a **UserManagement** per guardar la informació personal de l'usuari (email, nom i cognoms).
 - Crida a **FileManagement** per crear la carpeta arrel de l'usuari.
 - Finalment, envia un missatge asíncron a través de RabbitMQ a **Sync-Service** per generar l'estat inicial de sincronització (*snapshot*) amb la carpeta arrel.

4. Un cop finalitzat el procés, **UserAuthentication** retorna els tokens d'accés i de refresh per iniciar la sessió automàticament.

A.2 UC-02: Iniciar sessió

- **Descripció:** L'usuari inicia sessió amb el seu nom i contrasenya.
- **Actors:** Usuari.
- **Precondicions:** Tenir un compte vàlid.
- **Postcondicions:** Sessió activa.
- **Escenari principal:**
 1. El client envia el nom d'usuari i la contrasenya al servei **UserAuthentication**.
 2. El servei busca l'usuari a la base de dades a partir del seu nom.
 3. Si l'usuari existeix, compara la contrasenya rebuda amb la versió xifrada emmagatzemada.
 4. Si les credencials són correctes, genera un nou token d'accés (JWT) de curta durada i un token de refresh de llarga durada.
 5. Finalment, retorna els dos tokens al client per iniciar la sessió i mantenir-la activa.

A.3 UC-03: Actualitzar perfil

- **Descripció:** L'usuari modifica les seves dades personals (nom, cognoms i email).
- **Actors:** Usuari.
- **Precondicions:** Sessió iniciada.
- **Postcondicions:** Dades de perfil actualitzades a la base de dades.
- **Escenari principal:**
 1. El Gateway valida el token JWT i reenvia la petició a **UserManagement** amb l'ID de l'usuari.
 2. El servei busca l'usuari a la base de dades.
 3. Valida que el nou email no estigui en ús per un altre usuari.
 4. Si la validació és correcta, actualitza el nom, cognoms i email de l'usuari.
 5. Retorna les dades actualitzades.

A.4 UC-04: Cercar usuaris

- **Descripció:** Permet localitzar usuaris per nom d'usuari o correu electrònic.
- **Actors:** Usuari.
- **Precondicions:** Sessió iniciada.
- **Postcondicions:** Retorna una llista d'usuaris que coincideixen amb el criteri de cerca.
- **Escenari principal:**
 1. El Gateway valida el JWT i envia la petició de cerca a **UserManagement**.
 2. **UserManagement** fa una crida a **UserAuthentication** per buscar coincidències per nom d'usuari.
 3. Paral·lelament, **UserManagement** busca a la seva pròpia base de dades coincidències per email.
 4. Combina i retorna una llista d'usuaris (nom d'usuari i email) que compleixen el criteri de cerca.

A.5 UC-05: Llistar usuaris (administració)

- **Descripció:** Un administrador o superadministrador consulta la llista completa d'usuaris del sistema.
- **Actors:** Administrador, Superadministrador.
- **Precondicions:** Rol d'administrador o superadministrador.
- **Postcondicions:** Llista de tots els usuaris amb les seves dades.
- **Escenari principal:**
 1. El Gateway valida el JWT i el rol del sol·licitant, i reenvia la petició a **UserManagement**.
 2. **UserManagement** obté la llista de tots els usuaris de la seva base de dades.
 3. A continuació, fa una crida a **UserAuthentication** per obtenir les dades d'autenticació (nom d'usuari i rol) de cada usuari.
 4. Si el sol·licitant és Superadministrador, la informació retornada per **UserAuthentication** inclou també la contrasenya.
 5. Finalment, **UserManagement** combina la informació i retorna una llista completa amb els detalls de cada usuari.

A.6 UC-06: Actualitzar usuari (administració)

- **Descripció:** Un administrador o superadministrador modifica les dades d'un altre usuari, incloent el seu rol o contrasenya.
- **Actors:** Administrador, Superadministrador.
- **Precondicions:** Rol administratiu i respectar la jerarquia de permisos.
- **Postcondicions:** Dades de l'usuari objectiu actualitzades.
- **Escenari principal:**
 1. El Gateway valida el JWT i el rol, i envia la petició a **UserManagement**.
 2. **UserManagement** fa una crida a **UserAuthentication** per comprovar els rols tant del sol·licitant com de l'usuari a modificar.
 3. Es valida la jerarquia: un Administrador només pot modificar usuaris amb rol USER, mentre que un Superadministrador pot modificar qualsevol usuari excepte treure's a si mateix el rol de Superadministrador.
 4. **UserManagement** actualitza la informació personal (nom, email) a la seva base de dades.
 5. Simultàniament, fa una crida a **UserAuthentication** perquè actualitzi les dades d'autenticació (nom d'usuari, rol i, si s'escau, la contrasenya).

A.7 UC-07: Eliminar usuari (administració)

- **Descripció:** Un administrador o superadministrador inicia el procés d'eliminació completa d'un usuari.
- **Actors:** Administrador, Superadministrador.
- **Precondicions:** Rol administratiu i respectar la jerarquia de permisos.
- **Postcondicions:** S'inicia l'eliminació asíncrona de les dades de l'usuari a tots els serveis.
- **Escenari principal:**
 1. El Gateway valida el JWT i el rol, i reenvia la petició a **UserManagement**.
 2. **UserManagement** crida a **UserAuthentication** per verificar la jerarquia de rols (un Administrador només pot eliminar usuaris USER, un Superadministrador pot eliminar Administradors).
 3. Si es compleixen els permisos, **UserManagement** envia un missatge d'eliminació a una cua de RabbitMQ.
 4. Aquest missatge és rebut per **UserAuthentication**, que esborra les credencials de l'usuari i, al seu torn, envia un nou missatge de notificació d'eliminació.

5. Aquest segon missatge és consumit per la resta de microserveis (**FileManagement**, **FileSharing**, etc.), que procedeixen a eliminar de forma asíncrona totes les dades associades a l'usuari.

A.8 UC-08: Crear o pujar arxius

- **Descripció:** Pujada de fitxers o creació de carpetes.
- **Actors:** Usuari.
- **Precondicions:** Permís d'escriptura a la carpeta de destinació.
- **Postcondicions:** Nou element emmagatzemat i notificat.
- **Escenari principal:**
 1. El Gateway valida el token JWT i reenvia la petició a **FileManagement**.
 2. El servei consulta a **FileAccessControl** per verificar que l'usuari té permís d'escriptura (WRITE) a la carpeta de destinació.
 3. Si el permís és correcte, crea les metadades de l'arxiu (nom, mida, etc.) a la seva base de dades.
 4. Emmagatzema el contingut del fitxer al sistema d'arxius del servidor, utilitzant un ID únic com a nom.
 5. Sol·licita a **FileAccessControl** que assigni el permís de propietari (ADMIN) sobre el nou element a l'usuari que l'ha pujat.
 6. Finalment, envia un missatge asíncron a **SyncService** per notificar la creació i actualitzar els clients.

A.9 UC-09A: Renomenar un element

- **Descripció:** Canvia el nom d'un arxiu o carpeta existent.
- **Actors:** Usuari.
- **Precondicions:** Permís d'escriptura sobre l'element.
- **Postcondicions:** L'element té el nou nom assignat.
- **Escenari principal:**
 1. El Gateway valida el JWT i envia la sol·licitud a **FileManagement** amb el nou nom.
 2. **FileManagement** crida a **FileAccessControl** per verificar que l'usuari té permís d'escriptura (WRITE) sobre l'element.
 3. Actualitza el nom de l'element a la base de dades.

4. Envia un missatge d'actualització (update) a **SyncService** per notificar el canvi als clients.

A.10 UC-09B: Moure un element

- **Descripció:** Canvia la ubicació d'un arxiu o carpeta a una nova carpeta de destinació.
- **Actors:** Usuari.
- **Precondicions:** Permís d'escriptura sobre l'element a moure i sobre la carpeta de destinació.
- **Postcondicions:** L'element es troba a la nova ubicació.
- **Escenari principal:**
 1. El Gateway valida el JWT i envia la sol·licitud a **FileManagement** amb l'ID de l'element i de la destinació.
 2. **FileManagement** verifica a **FileAccessControl** el permís d'escriptura (WRITE) sobre l'origen i la destinació.
 3. Valida que l'operació no generi una dependència circular (p. ex., moure una carpeta dins d'ella mateixa).
 4. Actualitza la referència a la carpeta pare de l'element a la base de dades.
 5. Envia un missatge d'actualització (update) a **SyncService** per notificar el canvi als clients.

A.11 UC-09C: Copiar un element

- **Descripció:** Crea un duplicat d'un arxiu o carpeta en una ubicació de destinació.
- **Actors:** Usuari.
- **Precondicions:** Permís de lectura sobre l'element a copiar i d'escriptura sobre la carpeta de destinació.
- **Postcondicions:** Es crea una còpia de l'element a la destinació.
- **Escenari principal:**
 1. El Gateway valida el JWT i envia la sol·licitud a **FileManagement**.
 2. **FileManagement** verifica a **FileAccessControl** el permís de lectura (READ) sobre l'origen i d'escriptura (WRITE) sobre la destinació.
 3. Crea les noves entitats a la base de dades per a la còpia, assignant nous IDs.

4. Si és un fitxer, duplica el contingut físic al sistema d'emmagatzematge.
5. Sol·licita a **FileAccessControl** que assigni el permís de propietari (ADMIN) sobre el nou element a l'usuari.
6. Envia un missatge de creació (create) a **SyncService** per notificar el nou element als clients.

A.12 UC-10: Descarregar

- **Descripció:** Obtenir el contingut d'un arxiu o carpeta.
- **Actors:** Usuari.
- **Precondicions:** Permís de lectura sobre l'element sol·licitat.
- **Postcondicions:** Arxiu descarregat pel client.
- **Escenari principal:**
 1. El Gateway valida el JWT i reenvia la petició a **FileManagement**.
 2. Aquest consulta **FileAccessControl** per confirmar que l'usuari té permís de lectura (READ).
 3. Si els permisos són correctes, recupera el fitxer del sistema d'emmagatzematge. Si és una carpeta, la comprimeix en format ZIP.
 4. Retorna el contingut com un flux de dades (*stream*) perquè el client iniciï la descàrrega.

A.13 UC-11: Enviar a la paperera

- **Descripció:** Moure elements a la paperera.
- **Actors:** Usuari.
- **Precondicions:** Permís d'escriptura sobre l'element.
- **Postcondicions:** Element marcat com a eliminat i visible a la paperera.
- **Escenari principal:**
 1. El Gateway valida el JWT i envia la petició a **TrashService**.
 2. **TrashService** verifica a **FileAccessControl** que l'usuari té permís d'escriptura.
 3. Crida a **FileManagement** perquè marqui l'element i els seus descendents com a eliminats (sense esborrar-los físicament).
 4. Crea un registre a la seva pròpia base de dades (TrashRecord) per cada element mogut, emmagatzemant la data d'eliminació i de caducitat.

5. **FileManagement** notifica a **SyncService** el canvi d'estat per actualitzar els clients.

A.14 UC-12: Eliminar permanent

- **Descripció:** Esborrar definitivament un element de la paperera.
- **Actors:** Usuari.
- **Precondicions:** Element a la paperera i ser-ne el propietari.
- **Postcondicions:** Element eliminat de forma permanent.
- **Escenari principal:**
 1. El Gateway envia la petició a **TrashService**.
 2. El servei verifica que l'usuari és el propietari de l'element.
 3. Crida a **FileManagement** per esborrar l'arxiu físic i les seves metadades.
 4. Canvia l'estat del TrashRecord a PENDING_DELETION i inicia un procés de purga asíncron.
 5. A través de missatges per cua, ordena a **FileAccessControl** i **FileSharing** que eliminin totes les regles associades a l'element.
 6. Un cop confirmat per tots els serveis, el TrashRecord s'esborra.
 7. Es notifica a **SyncService** per eliminar les còpies locals de l'element.

A.15 UC-13: Compartir arxius

- **Descripció:** Concedir o revocar accessos sobre elements a altres usuaris.
- **Actors:** Usuari.
- **Precondicions:** Permís de propietari o d'administrador sobre l'element.
- **Postcondicions:** Els usuaris seleccionats obtenen o perden l'accés indicat.
- **Escenari principal:**
 1. El Gateway valida el JWT i passa la petició a **FileSharing**.
 2. El servei verifica a **FileAccessControl** que el sol·licitant és el propietari (ADMIN) de l'element.
 3. Consulta a **UserManagement** per obtenir l'ID de l'usuari amb qui es vol compartir.
 4. Sol·licita a **FileAccessControl** que creï una nova regla d'accés (lectura o escriptura) per a l'usuari convidat sobre l'element i els seus descendents (si es una carpeta).

5. Desa un registre de la compartició a la seva base de dades.
6. Notifica a **SyncService** a través de RabbitMQ per propagar els canvis als clients implicats.

A.16 UC-13A: Revocar accés a un arxiu (proprietari)

- **Descripció:** El propietari d'un element compartit revoca el permís d'accés a un altre usuari.
- **Actors:** Usuari (proprietari).
- **Precondicions:** Ser el propietari (ADMIN) de l'element.
- **Postcondicions:** L'usuari objectiu perd l'accés a l'element.
- **Escenari principal:**
 1. El Gateway valida el JWT i reenvia la petició a **FileSharing** amb l'ID de l'element i el nom de l'usuari a qui es revoca el permís.
 2. **FileSharing** crida a **FileAccessControl** per verificar que el sol·licitant és el propietari (ADMIN) de l'element.
 3. Sol·licita a **UserManagement** l'ID de l'usuari a eliminar.
 4. Crida a **FileAccessControl** per eliminar la regla d'accés associada a l'usuari i a l'element (i als seus descendents, si és una carpeta).
 5. Elimina el registre corresponent de la taula de SharedAccess de la seva pròpia base de dades.
 6. Notifica a **SyncService** a través de RabbitMQ per actualitzar els clients implicats.

A.17 UC-13B: Deixar de seguir un arxiu compartit (receptor)

- **Descripció:** Un usuari elimina un element que algú altre havia compartit amb ell.
- **Actors:** Usuari (receptor).
- **Precondicions:** Un altre usuari ha compartit un element amb l'usuari actual.
- **Postcondicions:** L'element desapareix de la llista d'arxius compartits de l'usuari.
- **Escenari principal:**
 1. El Gateway valida el JWT i reenvia la petició a **FileSharing**. El flux és idèntic a UC-13A, però en aquest cas el sol·licitant és el mateix usuari a qui es revocarà el permís.

2. **FileSharing** comprova que el sol·licitant i l'usuari a eliminar són el mateix.
3. Es crida a **FileAccessControl** per eliminar la regla d'accés i a la base de dades de **FileSharing** per eliminar el registre de compartició.
4. Finalment, es notifica a **SyncService** per actualitzar la interfície de l'usuari.

A.18 UC-14: Actualització en temps real

- **Descripció:** Manté els clients sincronitzats amb els canvis del sistema de fitxers mitjançant WebSockets.
- **Actors:** Usuari.
- **Precondicions:** Sessió iniciada.
- **Postcondicions:** El client rep esdeveniments de sincronització i actualitza el seu estat local.
- **Escenari principal:**
 1. El client (Tauri o web) estableix una connexió WebSocket amb el Gateway, enviant el token JWT a les capçaleres.
 2. El Gateway valida el token i reenvia la connexió a **SyncService**, afegint a les capçaleres l'ID d'usuari i un ID de connexió únic.
 3. **SyncService** registra la connexió WebSocket associada a l'usuari. Si el client és d'escriptori (Tauri), li envia l'estat actual complet del seu arbre de fitxers (*snapshot*).
 4. Quan un altre servei (com **FileManagement** o **FileSharing**) realitza una acció que modifica l'estructura de fitxers, envia un missatge a una cua de RabbitMQ.
 5. **SyncService** consumeix aquest missatge, actualitza l'snapshot de l'usuari afectat i propaga el canvi.
 6. Per als clients d'escriptori, envia el *snapshot* actualitzat. Per als clients web, envia una ordre de refresh (updated_tree) per a la part de la interfície afectada. Això es fa a través de la connexió WebSocket corresponent.

A.19 UC-15: Canviar contrasenya

- **Descripció:** L'usuari actualitza la seva contrasenya.
- **Actors:** Usuari.
- **Precondicions:** Sessió iniciada i coneixement de la contrasenya actual.
- **Postcondicions:** Contrasenya modificada a la base de dades.

- Escenari principal:

1. El Gateway valida el JWT i reenvia la petició a **UserAuthentication** amb l'ID de l'usuari i les contrasenyes (antiga i nova).
2. El servei recupera l'usuari de la base de dades.
3. Compara la contrasenya antiga rebuda amb el que hi ha emmagatzemat per verificar-la.
4. Valida que la nova contrasenya compleix els requisits de seguretat (llargada, majúscules, minúscules i números).
5. Si tot és correcte, xifra la nova contrasenya i l'actualitza a la base de dades.

A.20 UC-16: Eliminar compte

- **Descripció:** L'usuari sol·licita esborrar definitivament el seu propi compte i tots els seus arxius.
- **Actors:** Usuari.
- **Precondicions:** Sessió iniciada.
- **Postcondicions:** S'inicia el procés d'eliminació asíncrona de totes les dades de l'usuari.
- **Escenari principal:**

1. El Gateway valida el JWT i passa la petició a **UserAuthentication**.
2. **UserAuthentication** esborra l'entitat de l'usuari de la seva base de dades.
3. A continuació, envia un missatge a una cua de RabbitMQ per notificar que un usuari ha estat eliminat.
4. La resta de microserveis (**UserManagement**, **FileManagement**, etc.) estan subscriptos a aquesta cua i, en rebre el missatge, cadascun inicia la purga de totes les dades associades a l'ID d'aquell usuari.

A.21 UC-17: Restaurar des de la paperera

- **Descripció:** Recuperar un element prèviament enviat a la paperera.
- **Actors:** Usuari.
- **Precondicions:** L'element es troba a la paperera i l'usuari n'és el propietari.
- **Postcondicions:** Element restaurat a la seva ubicació original i ja no és visible a la paperera.
- **Escenari principal:**

1. El Gateway valida el JWT i envia la sol·licitud a **TrashService**.
2. **TrashService** verifica que l'usuari que fa la petició és el propietari.
3. Crida a **FileManagement** per canviar l'estat de l'element (i els seus descendents, si és una carpeta) a no eliminat.
4. **TrashService** elimina de la seva base de dades tots els registres (TrashRecord) associats als elements restaurats.
5. Finalment, **FileManagement** notifica a **SyncService** el canvi d'estat a través de RabbitMQ per actualitzar la vista dels clients.

A.22 UC-18: Modificar contrasenya d'un altre usuari (Superadministració)

- **Descripció:** El Superadministrador canvia la contrasenya d'un altre usuari.
- **Actors:** Superadministrador.
- **Precondicions:** Rol de Superadministrador.
- **Postcondicions:** Contrasenya de l'usuari objectiu modificada.
- **Escenari principal:**
 1. La petició arriba a **UserManagement** a través del Gateway.
 2. **UserManagement** crida a **UserAuthentication** per validar que el sol·licitant és Superadministrador.
 3. Si la validació és correcta, **UserManagement** envia una ordre d'actualització a **UserAuthentication** amb la nova contrasenya.
 4. **UserAuthentication** valida el format de la nova contrasenya, la xifra i l'emmagatzema a la seva base de dades.

A.23 UC-19: Modificar un administrador (Superadministració)

- **Descripció:** El Superadministrador modifica les dades d'un usuari amb rol d'Administrador.
- **Actors:** Superadministrador.
- **Precondicions:** Rol de Superadministrador.
- **Postcondicions:** Dades de l'administrador actualitzades.
- **Escenari principal:**
 1. Similar a UC-06, el Gateway reenvia la petició a **UserManagement**.

2. **UserManagement** crida a **UserAuthentication** per verificar que el sol·licitant és SUPER_ADMIN i l'usuari a modificar és ADMIN.
3. Si la jerarquia de permisos és correcta, **UserManagement** actualitza la informació personal.
4. Paral·lelament, **UserAuthentication** actualitza les dades d'autenticació (nom d'usuari, rol).

A.24 UC-20: Eliminar un administrador (Superadministració)

- **Descripció:** El Superadministrador elimina un compte d'usuari amb rol d'Administrador.
- **Actors:** Superadministrador.
- **Precondicions:** Rol de Superadministrador.
- **Postcondicions:** S'inicia l'eliminació asíncrona de les dades de l'administrador.
- **Escenari principal:**
 1. El Gateway valida el rol i reenvia la petició a **UserManagement**.
 2. **UserManagement** crida a **UserAuthentication** per verificar que el sol·licitant és SUPER_ADMIN i l'usuari a eliminar és ADMIN.
 3. Si els permisos són correctes, s'inicia el mateix procés d'eliminació asíncrona descrit a UC-07, enviant missatges a la resta de serveis per purgar totes les dades associades.

A.25 UC-21: Modificar nivell de permisos d'un usuari (Superadministració)

- **Descripció:** El Superadministrador canvia el rol d'un usuari (p. ex., de USER a ADMIN).
- **Actors:** Superadministrador.
- **Precondicions:** Rol de Superadministrador.
- **Postcondicions:** El rol de l'usuari objectiu és modificat.
- **Escenari principal:**
 1. El flux és una variant de UC-06. La petició arriba a **UserManagement**.
 2. Es verifica a **UserAuthentication** que el sol·licitant té permisos per realitzar el canvi de rol. Un Superadministrador no pot rebaixar el seu propi rol.

3. **UserManagement** envia la petició a **UserAuthentication** per actualitzar el camp role de l'usuari a la base de dades d'autenticació.

A.26 UC-22: Sincronitzar arxius compartits

- **Descripció:** Manté els clients sincronitzats amb els canvis en arxius i carpetes que altres usuaris han compartit amb ells. Aquesta funcionalitat no va ser completament implementada.
- **Actors:** Usuari.
- **Precondicions:** Altres usuaris han compartit elements amb l'usuari actual.
- **Postcondicions:** El client web rep notificacions sobre canvis en elements compartits.
- **Escenari principal:**
 1. Quan un usuari propietari comparteix un element (a la UC-13), el servei **FileSharing** envia un missatge a RabbitMQ.
 2. **SyncService** rep aquest missatge.
 3. Per al client web, **SyncService** envia una ordre genèrica de refresh (updated_tree) a l'usuari amb qui s'ha compartit l'element, indicant-li que ha de refrescar la secció d'arxius compartits.
 4. Per al client d'escriptori (Tauri), aquesta funcionalitat no està implementada. El disseny preveia que **SyncService** actualitzés un *snapshot* específic per als elements compartits, però no es va dur a terme.

Apèndix B

Diagrams d'activitat de tots els Casos d'Ús

A continuació, es presenten els diagrames d'activitat detallats per a cada un dels casos d'ús definits en el sistema. Cada diagrama il·lustra el flux de treball, les decisions i les interaccions entre serveis.

UC-01: Registrar-se

El flux comença quan l'usuari envia el formulari de registre. El Gateway reenvia la petició a UserAuthentication, que valida el format de les dades, comprova que el nom d'usuari no existeix i consulta a UserManagement per assegurar que l'email tampoc estigui en ús. Si tot és correcte, orquestra la creació de l'usuari: guarda credencials, demana a UserManagement que creï el perfil, a FileManagement que generi la carpeta arrel i notifica a SyncService via RabbitMQ. Finalment, retorna els tokens per iniciar la sessió.

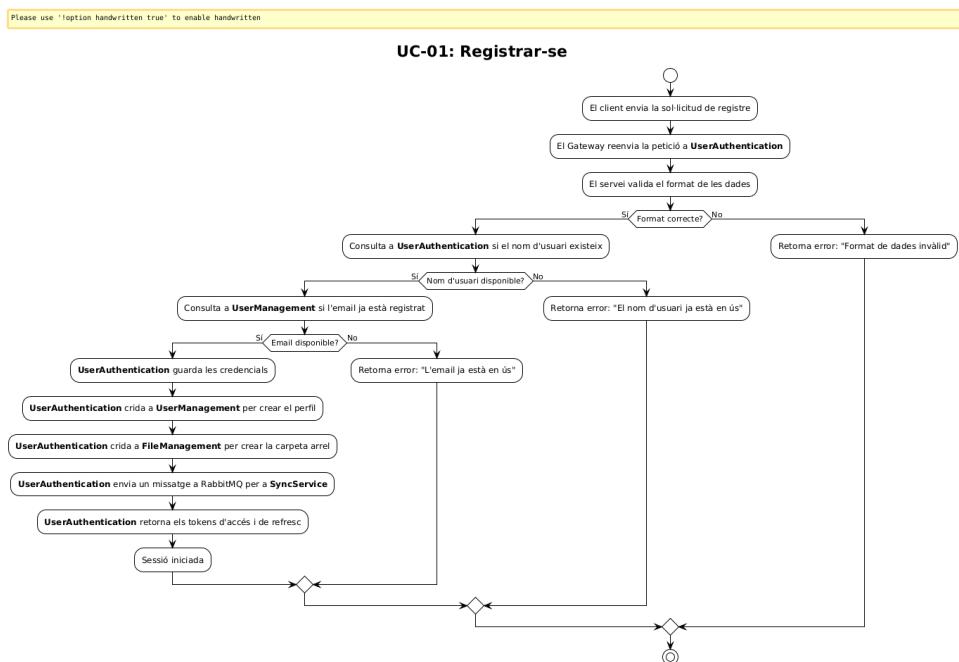


FIGURA B.1: Diagrama d'activitat per al cas d'ús UC-01: Registrar-se.

UC-02: Iniciar sessió

L'usuari envia les seves credencials, que el Gateway reenvia a UserAuthentication. El servei busca l'usuari i, si existeix, verifica la contrasenya. Si les credencials són correctes, genera i retorna un nou joc de tokens (accés i refresh) per activar la sessió del client.

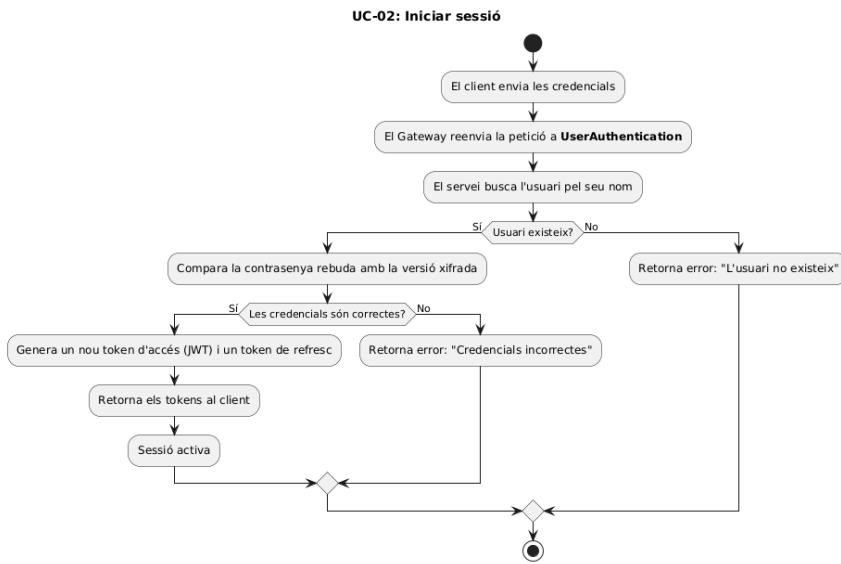


FIGURA B.2: Diagrama d'activitat per al cas d'ús UC-02: Iniciar sessió.

UC-03: Actualitzar perfil

L'usuari envia les seves dades de perfil actualitzades. El Gateway reenvia la petició a UserManagement, que valida que el nou correu electrònic no estigui ja en ús per un altre compte. Si la validació és correcta, actualitza les dades a la base de dades i retorna la informació actualitzada.

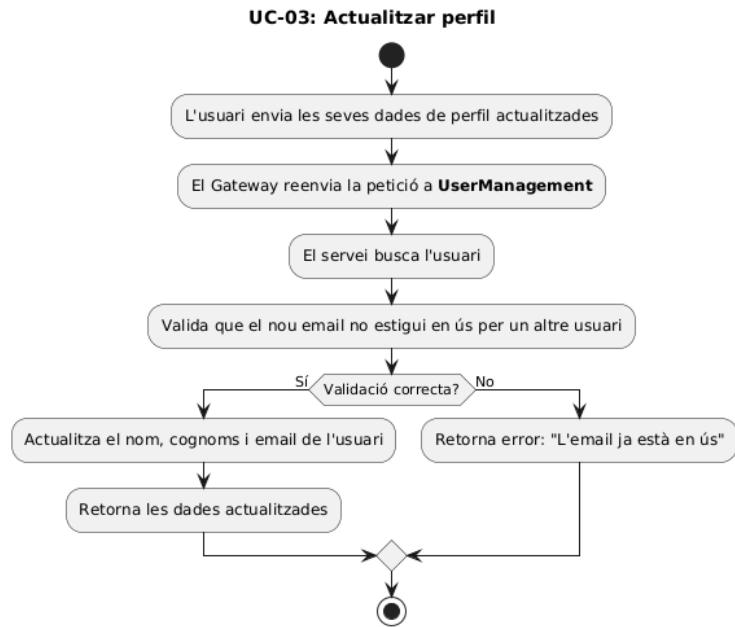


FIGURA B.3: Diagrama d'activitat per al cas d'ús UC-03: Actualitzar perfil.

UC-04: Cercar usuaris

L'usuari introduceix un terme de cerca. UserManagement rep la petició i llança dues cerques en paral·lel: una contra UserAuthentication per nom d'usuari i una altra a la seva pròpia base de dades per correu electrònic. Finalment, combina els resultats i els retorna.

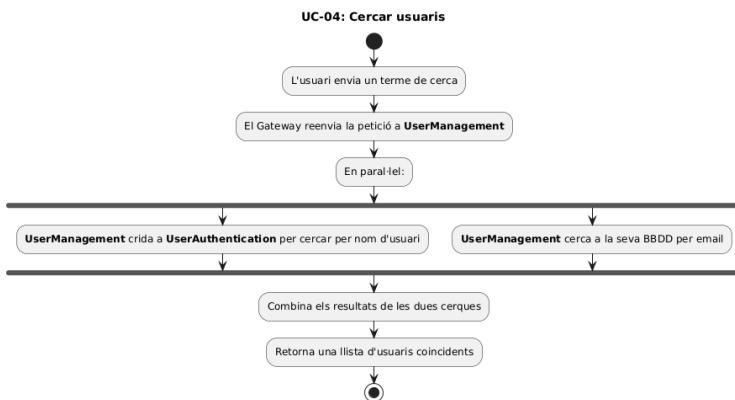


FIGURA B.4: Diagrama d'activitat per al cas d'ús UC-04: Cercar usuaris.

UC-05: Llistar usuaris (administració)

Un administrador sol·licita la llista completa d'usuaris. UserManagement obté la informació bàsica de la seva base de dades i la complementa amb les dades d'autenticació (rol, nom d'usuari) obtingudes de UserAuthentication. Si el sol·licitant és Superadministrador, la resposta inclou també les contrasenyes.

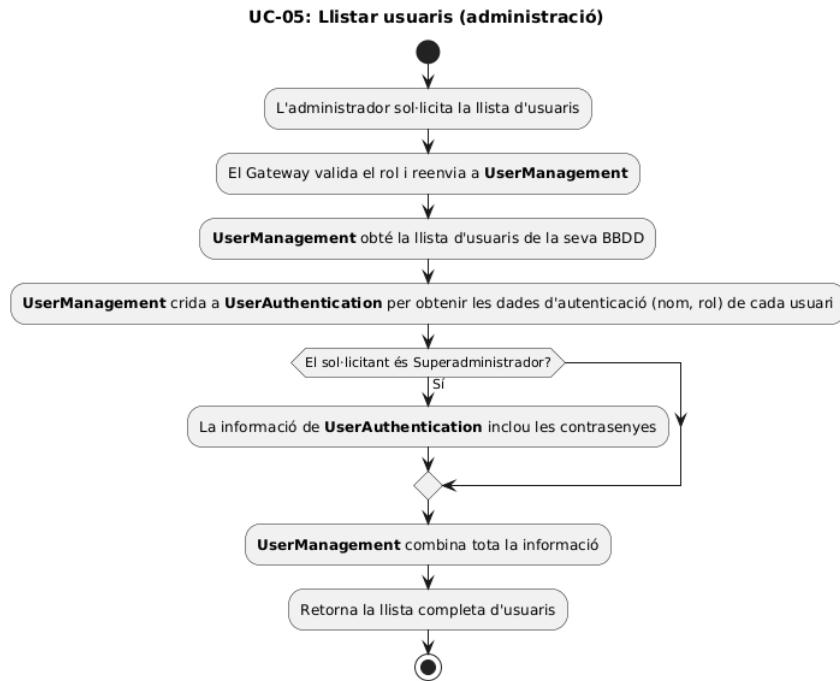


FIGURA B.5: Diagrama d'activitat per al cas d'ús UC-05: Llistar usuaris.

UC-06: Actualitzar usuari (administració)

Un administrador modifica les dades d'un altre usuari. UserManagement valida primer la jerarquia de permisos consultant a UserAuthentication. Si l'acció és permesa, actualitza la informació personal i, en paral·lel, demana a UserAuthentication que actualitzi les dades d'autenticació (rol o contrasenya).

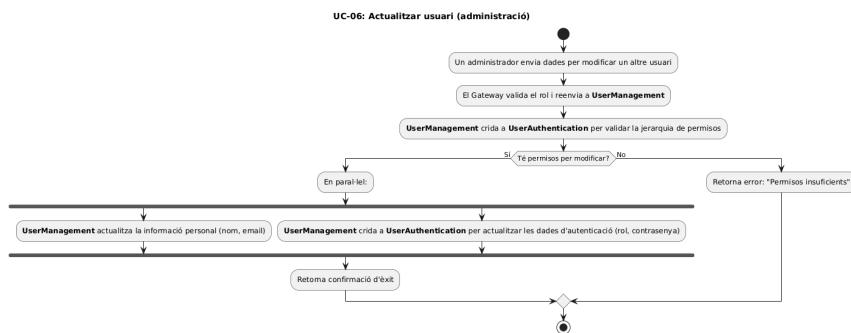


FIGURA B.6: Diagrama d'activitat per al cas d'ús UC-06: Actualitzar usuari.

UC-07: Eliminar usuari (administració)

Després de verificar la jerarquia de permisos amb UserAuthentication, UserManagement inicia el procés d'eliminació enviant un missatge a una cua de RabbitMQ. Això desencadena la purga asíncrona de les dades de l'usuari a tots els serveis del sistema.

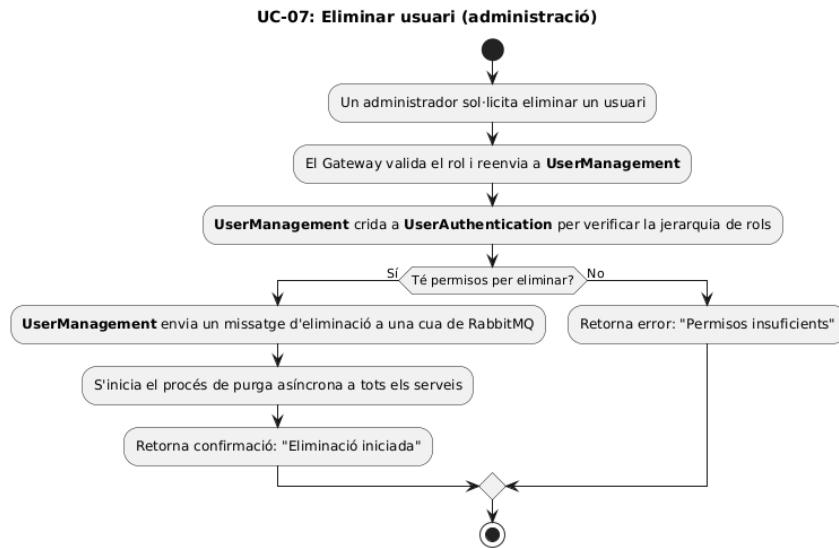


FIGURA B.7: Diagrama d'activitat per al cas d'ús UC-07: Eliminar usuari.

UC-08: Crear o pujar arxius

FileManagement rep la petició i consulta a FileAccessControl si l'usuari té permís d'escriptura. Si és així, crea les metadades, emmagatzema el fitxer (si escau), demana a FileAccessControl que assigni el permís de propietari i finalment notifica SyncService del canvi.

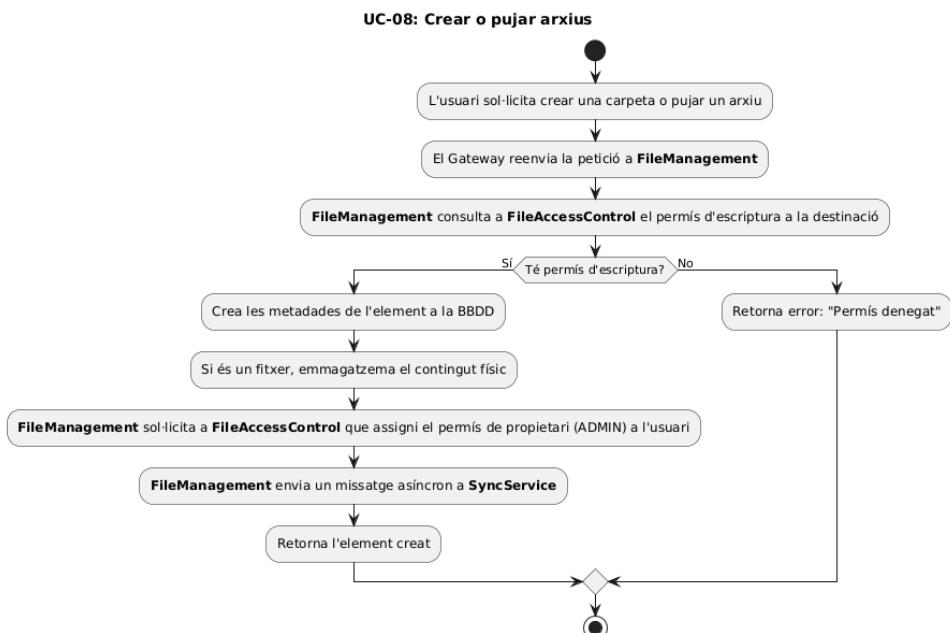


FIGURA B.8: Diagrama d'activitat per al cas d'ús UC-08: Crear o pujar arxius.

UC-09A: Renomenar un element

FileManagement verifica el permís d'escriptura a FileAccessControl. Si l'usuari té accés, actualitza el nom a la seva base de dades i envia un missatge a SyncService per notificar el canvi als clients.

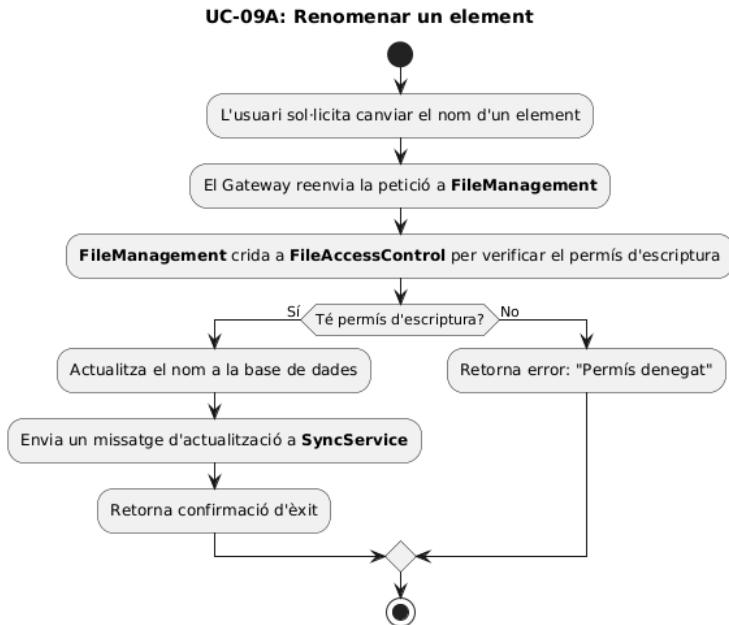


FIGURA B.9: Diagrama d'activitat per al cas d'ús UC-09A: Renomenar.

UC-09B: Moure un element

El servei FileManagement verifica els permisos d'escriptura tant a l'element d'origen com a la carpeta de destinació. A més, valida que l'operació no creï una dependència circular (moure una carpeta dins d'ella mateixa). Si tot és correcte, actualitza la ubicació i notifica SyncService.

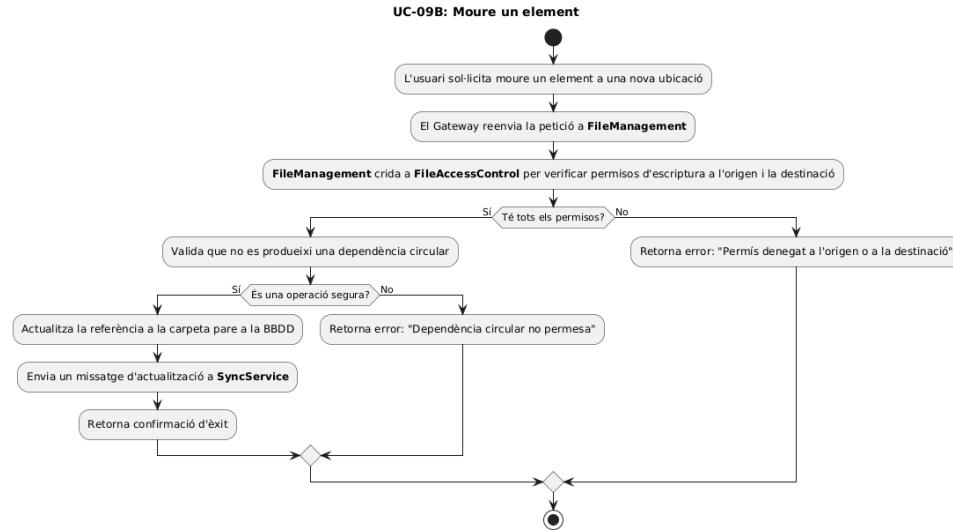


FIGURA B.10: Diagrama d'activitat per al cas d'ús UC-09B: Moure.

UC-09C: Copiar un element

FileManagement comprova el permís de lectura a l'origen i d'escriptura a la destinació. Si es compleixen, duplica les metadades i el contingut físic (si és un fitxer), assigna el permís de propietari sobre la còpia a través de FileAccessControl i notifica SyncService.

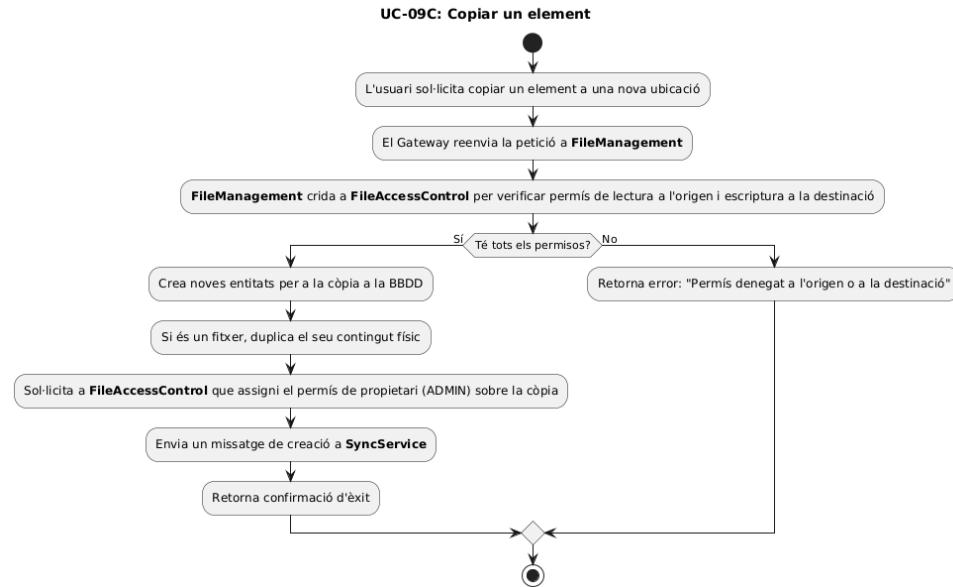


FIGURA B.11: Diagrama d'activitat per al cas d'ús UC-09C: Copiar.

UC-10: Descarregar

Després de verificar el permís de lectura a FileAccessControl, FileManagement recupera el fitxer del sistema d'emmagatzematge (o el comprimeix si és una carpeta) i el retorna al client com un flux de dades.

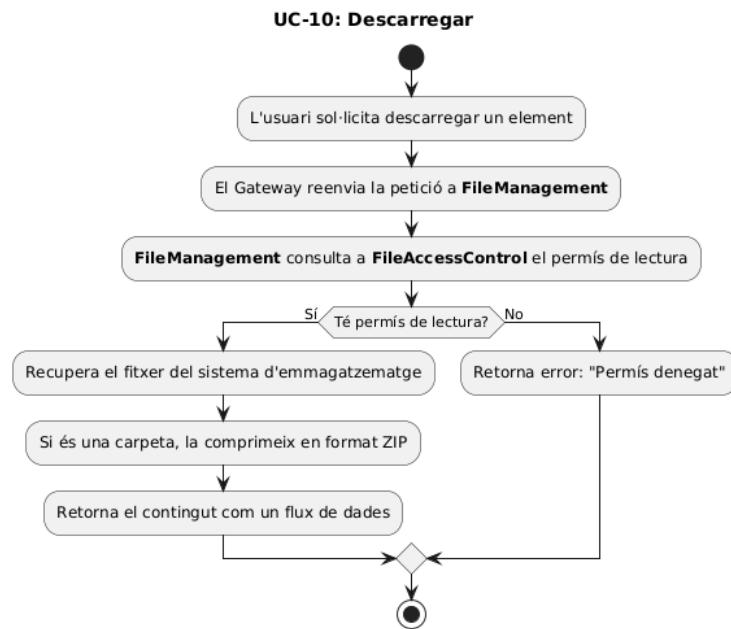


FIGURA B.12: Diagrama d'activitat per al cas d'ús UC-10: Descarregar.

UC-11: Enviar a la paperera

TrashService rep la petició, verifica el permís d'escriptura a FileAccessControl i demana a FileManagement que marqui l'element com a eliminat. Finalment, crea un registre a la seva pròpia base de dades per gestionar la caducitat de l'element.

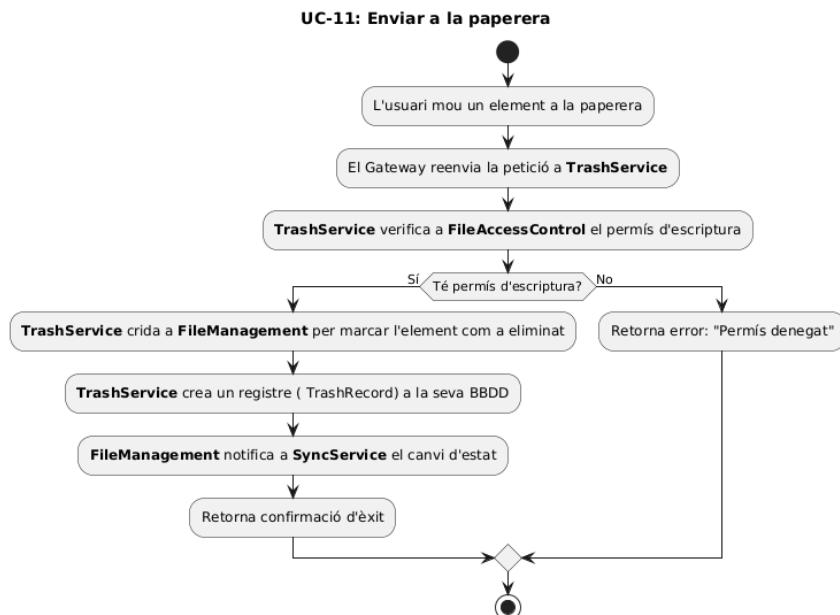


FIGURA B.13: Diagrama d'activitat per al cas d'ús UC-11: Enviar a la paperera.

UC-12: Eliminar permanentment

Quan un usuari sol·licita l'eliminació permanent, TrashService verifica que n'és el propietari. Si ho és, inicia la saga d'eliminació enviant missatges a la cua perquè FileManagement, FileAccessControl i FileSharing purguin totes les dades associades de forma asíncrona.

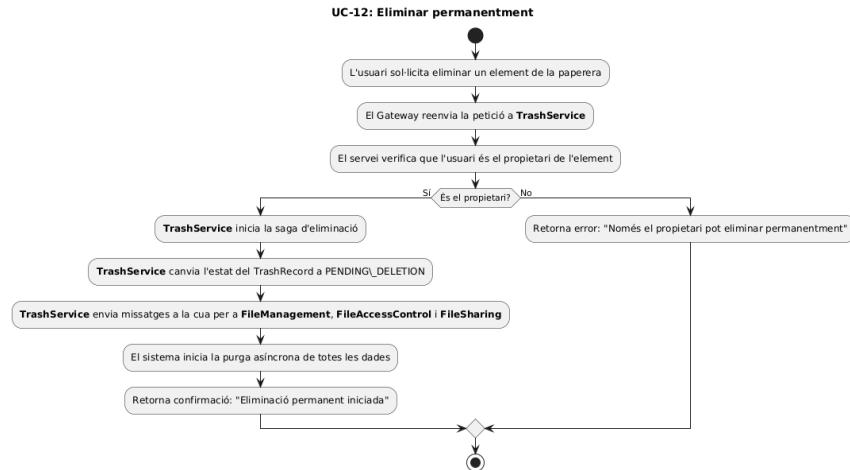


FIGURA B.14: Diagrama d'activitat per al cas d'ús UC-12: Eliminar permanentment.

UC-13: Compartir arxius

El servei FileSharing comprova que el sol·licitant és el propietari de l'element. Després, obté l'ID de l'usuari convidat de UserManagement i demana a FileAccessControl que creï la nova regla d'accés. Finalment, desa un registre de la compartició i notifica SyncService.

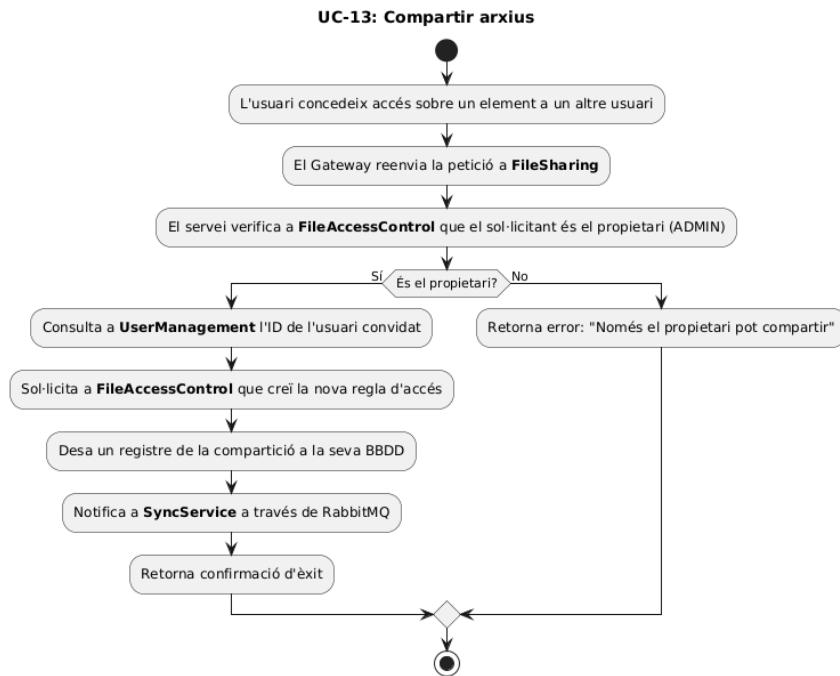


FIGURA B.15: Diagrama d'activitat per al cas d'ús UC-13: Compartir arxius.

UC-13A: Revocar accés a un arxiu

El propietari d'un element revoca l'accés a un altre usuari. FileSharing verifica la propietat, demana a FileAccessControl que elimini la regla de permís corresponent, esborra el registre de la seva base de dades i notifica el canvi a SyncService.

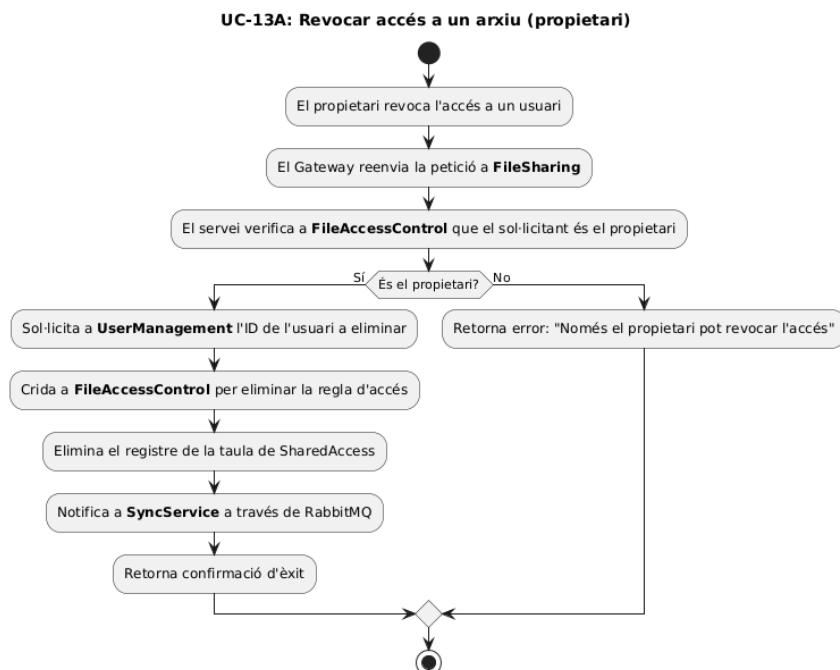


FIGURA B.16: Diagrama d'activitat per al cas d'ús UC-13A: Revocar accés.

UC-13B: Deixar de seguir un arxiu

Un usuari receptor decideix eliminar un element que li han compartit. El flux és gairebé idèntic a la revocació, però la validació inicial comprova que el sol·licitant és el mateix usuari que perdrà l'accés. FileSharing elimina la regla d'accés i el registre de compartició.

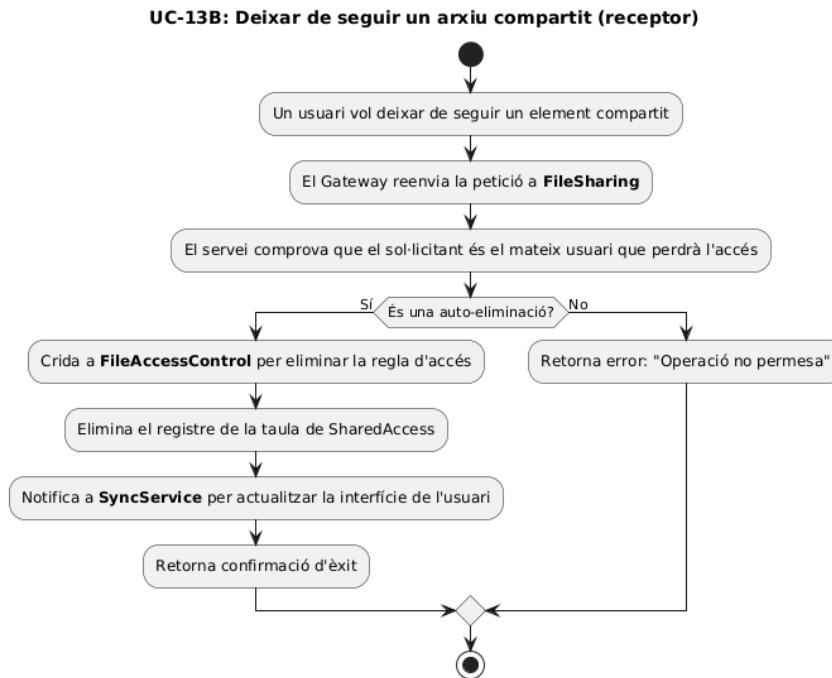


FIGURA B.17: Diagrama d'activitat per al cas d'ús UC-13B: Deixar de seguir.

UC-14: Actualització en temps real

El client estableix una connexió WebSocket amb SyncService a través del Gateway. Quan un altre servei publica un canvi a RabbitMQ, SyncService consumeix el missatge, actualitza l'estat intern de l'usuari afectat i li envia la notificació corresponent a través de la connexió oberta.

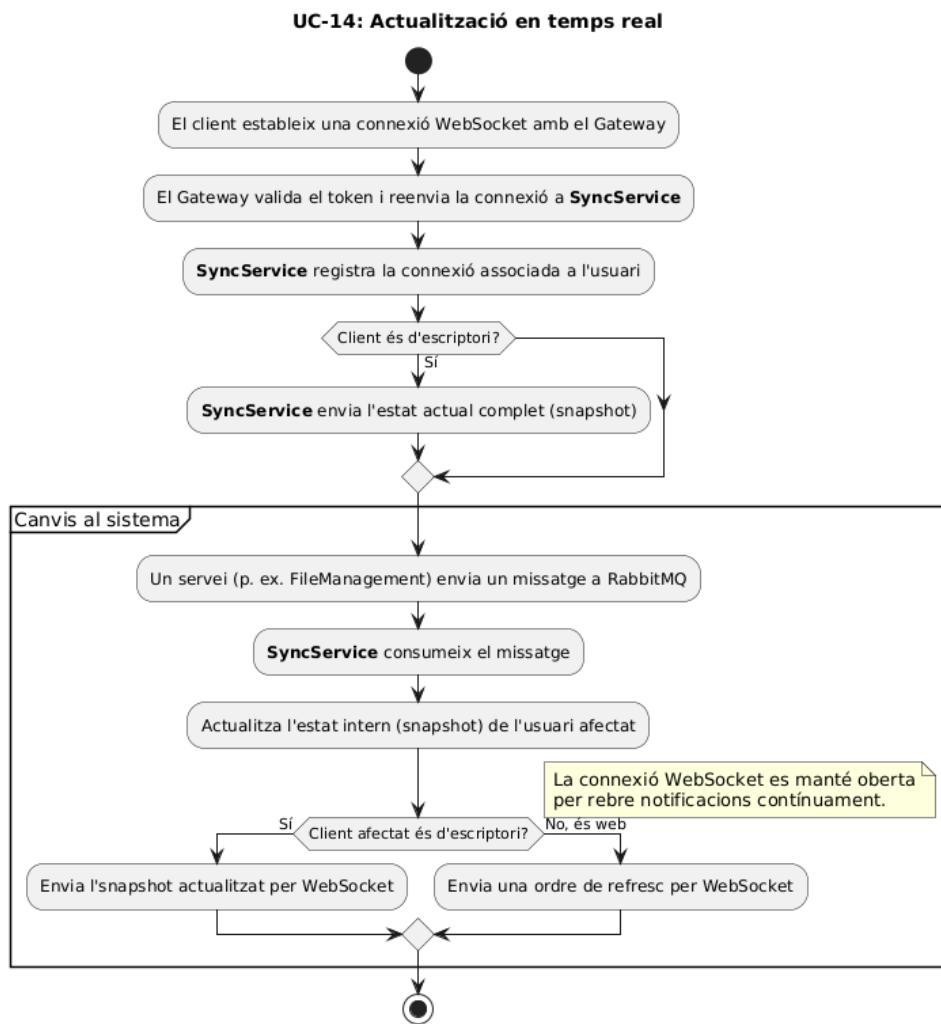


FIGURA B.18: Diagrama d'activitat per al cas d'ús UC-14: Actualització en temps real.

UC-15: Canviar contrasenya

UserAuthentication rep la petició, verifica que la contrasenya antiga sigui correcta i, si és així, valida que la nova compleixi els requisits de seguretat. Si tot és correcte, la xifra i l'actualitza a la base de dades.

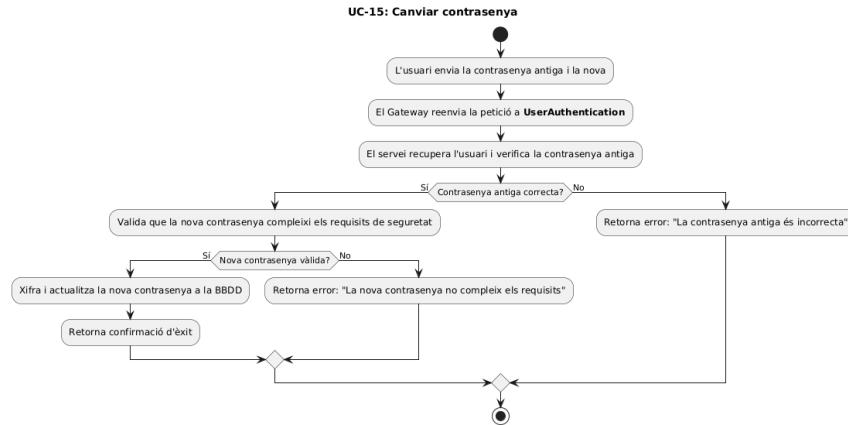


FIGURA B.19: Diagrama d'activitat per al cas d'ús UC-15: Canviar contrasenya.

UC-16: Eliminar compte

L'usuari sol·licita eliminar el seu propi compte. UserAuthentication esborra l'entitat d'autenticació i immediatament envia un missatge a RabbitMQ, iniciant el procés de purga asíncrona de totes les dades de l'usuari a la resta de serveis.

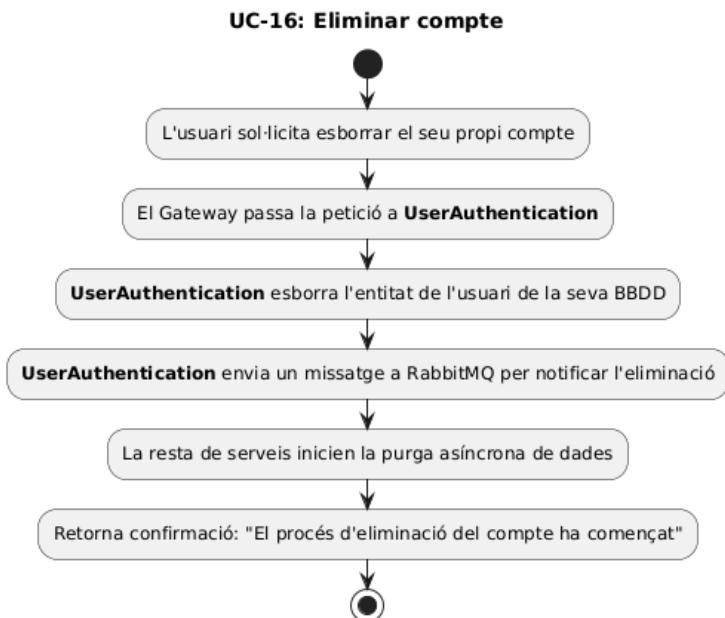


FIGURA B.20: Diagrama d'activitat per al cas d'ús UC-16: Eliminar compte.

UC-17: Restaurar des de la paperera

TrashService verifica que el sol·licitant és el propietari. Si ho és, demana a FileManagement que restableixi l'estat de l'element, elimina el registre de la seva pròpia base de dades i notifica SyncService perquè els clients actualitzin la seva vista.

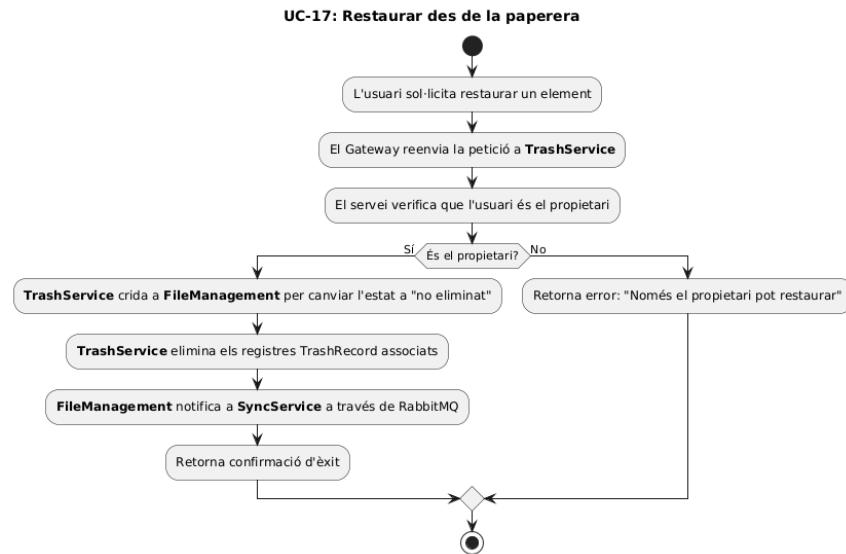


FIGURA B.21: Diagrama d'activitat per al cas d'ús UC-17: Restaurar des de la paperera.

UC-18: Modificar contrasenya d'un altre usuari

Un Superadministrador canvia la contrasenya d'un altre usuari. UserManagement valida el rol del sol·licitant i, si té permís, envia l'ordre a UserAuthentication, que valida el format de la nova contrasenya, la xifra i la desa.

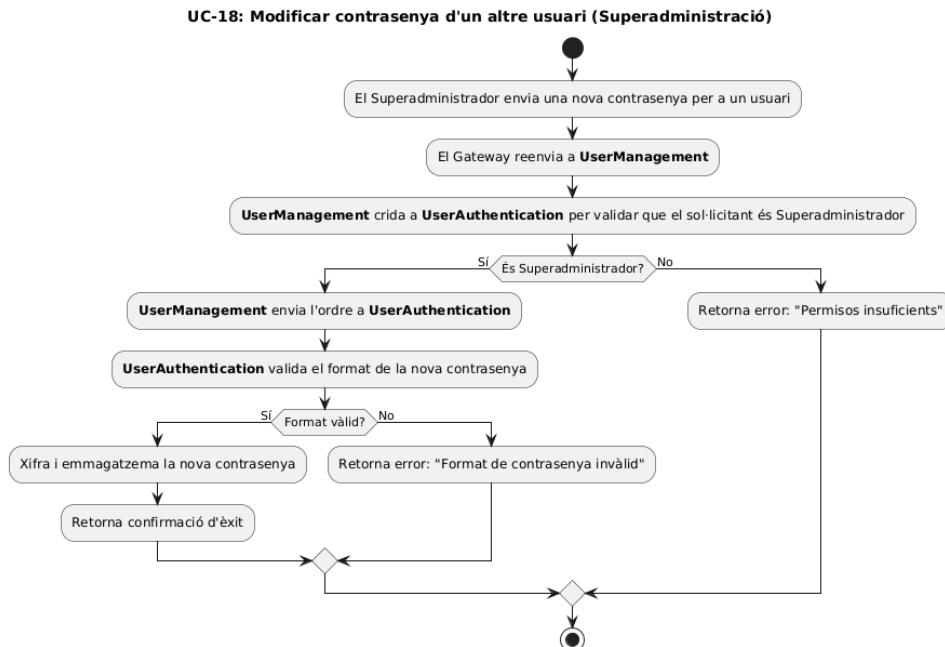


FIGURA B.22: Diagrama d'activitat per al cas d'ús UC-18: Modificar contrasenya d'altri.

UC-19: Modificar un administrador

El Superadministrador modifica un usuari amb rol d'Administrador. El flux és similar a l'UC-06, però la validació de jerarquia a UserAuthentication comprova específicament que un Superadministrador estigui modificant un Administrador.

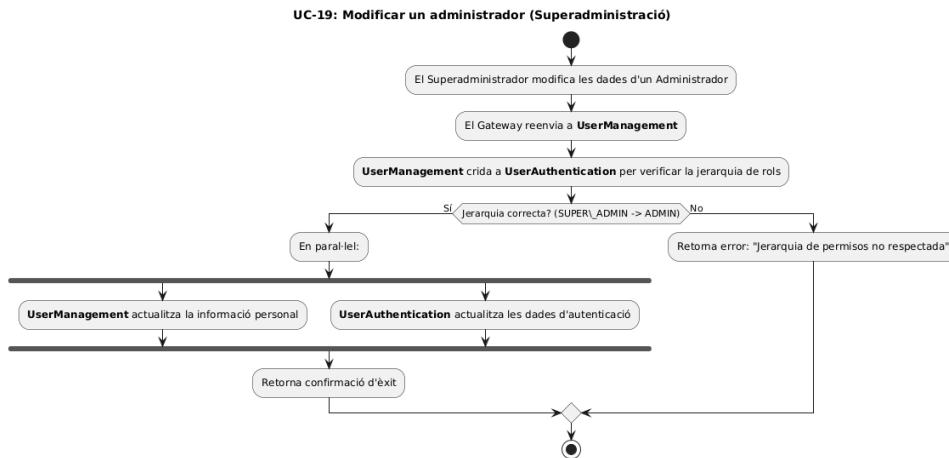


FIGURA B.23: Diagrama d'activitat per al cas d'ús UC-19: Modificar un administrador.

UC-20: Eliminar un administrador

El flux és idèntic a l'UC-07, però la comprovació de permisos que realitza UserManagement amb UserAuthentication valida que un Superadministrador estigui eliminant un usuari amb rol d'Administrador.

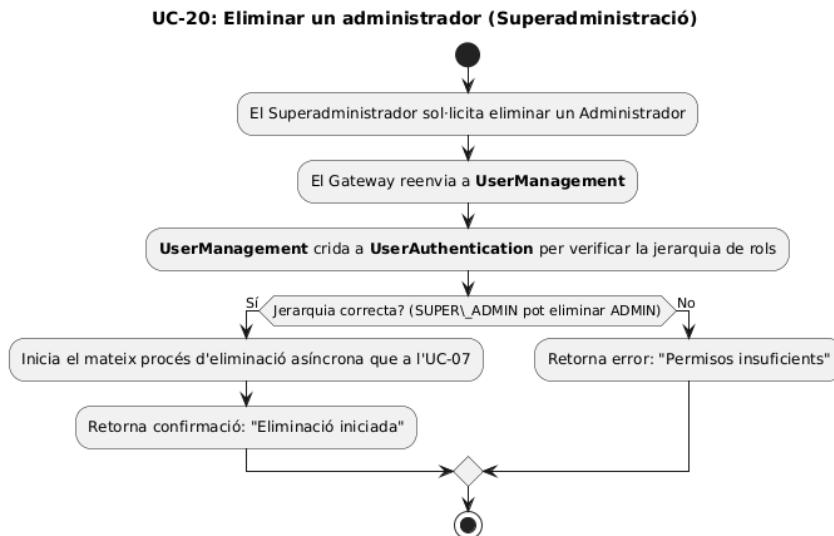


FIGURA B.24: Diagrama d'activitat per al cas d'ús UC-20: Eliminar un administrador.

UC-21: Modificar rol d'un usuari

El Superadministrador canvia el nivell de permisos d'un usuari. UserManagement demana a UserAuthentication que validi si l'operació és permesa (p. ex., un Superadministrador no pot rebaixar-se el seu propi rol). Si és vàlid, UserAuthentication actualitza el rol a la base de dades.

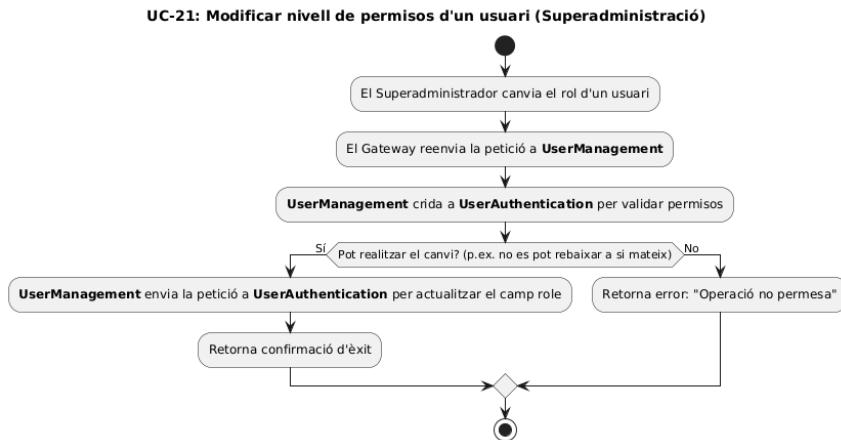


FIGURA B.25: Diagrama d'activitat per al cas d'ús UC-21: Modificar rol d'usuari.

UC-22: Sincronitzar arxius compartits

Aquest flux es desencadena quan un usuari en comparteix un altre (UC-13). FileSharing envia un missatge a RabbitMQ. SyncService el rep i, si el client del receptor és web, li envia una ordre genèrica de refresh. La funcionalitat completa per al client d'escriptori no es va arribar a implementar.

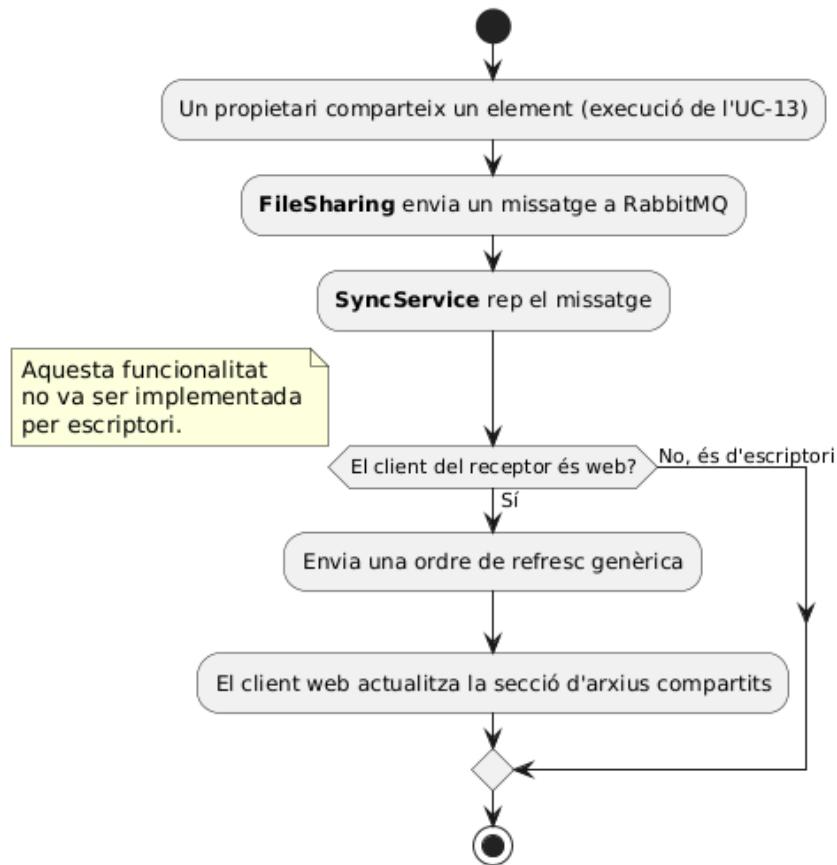
UC-22: Sincronitzar arxius compartits

FIGURA B.26: Diagrama d'activitat per al cas d'ús UC-22: Sincronitzar arxius compartits.

Bibliografia

- [1] Baeldung. *Why Choose Spring as Your Java Framework?* URL: <https://www.baeldung.com/spring-why-to-choose> (cons. 15-11-2023).
- [2] DB-Engines. *System Properties Comparison MySQL vs. Oracle vs. PostgreSQL.* URL: <https://db-engines.com/en/system/MySQL;Oracle;PostgreSQL> (cons. 15-11-2023).
- [3] CloudAMQP. *When to use RabbitMQ or Apache Kafka.* 2023. URL: <https://www.cloudamqp.com/blog/when-to-use-rabbitmq-or-apache-kafka.html> (cons. 15-11-2023).
- [4] J. Holcombe. *Estadísticas Clave de GitHub en 2025 (Usuarios, Empleados y Tendencias).* 2025. URL: <https://kinsta.com/es/blog/github-estadisticas/>.
- [5] *Spring Boot Reference Guide.* 2023. URL: <https://spring.io/projects/spring-boot>.
- [6] *React Documentation.* 2023. URL: <https://react.dev>.
- [7] *Tauri Documentation.* 2023. URL: <https://tauri.app>.
- [8] *Svelte Documentation.* 2023. URL: <https://svelte.dev>.
- [9] *Docker Documentation.* 2023. URL: <https://docs.docker.com>.
- [10] *PostgreSQL Documentation.* 2023. URL: <https://www.postgresql.org/docs/>.
- [11] *RabbitMQ Tutorials.* 2023. URL: <https://www.rabbitmq.com>.
- [12] *Radix UI Primitives.* 2023. URL: <https://www.radix-ui.com/primitives/docs>.
- [13] *Tremor React Components.* 2023. URL: <https://www.tremor.so>.
- [14] *TanStack React Query.* 2023. URL: <https://tanstack.com/query>.
- [15] *Zustand State Manager.* 2023. URL: <https://github.com/pmndrs/zustand>.
- [16] *dnd-kit Documentation.* 2023. URL: <https://dndkit.com>.
- [17] *Selecto Library.* 2023. URL: <https://daybrush.com/selecto/>.
- [18] *Remix Icon Library.* 2023. URL: <https://remixicon.com>.
- [19] *useDebounce Hook - usehooks-ts.* 2023. URL: <https://usehooks-ts.com/react-hook/use-debounce>.
- [20] Maggie Appleton. *What the Fork is the React Virtual DOM.* URL: <https://maggieappleton.com/react-vdom/>.
- [21] William Jones. *Conceptualizing React Components through Data flow Diagrams.* URL: <https://medium.com/@williamjonescodes/conceptualizing-react-through-data-flow-diagrams-91d0518d3d59>.
- [22] *React Visualized.* URL: <https://react.gg/visualized>.
- [23] Open Source Initiative. *MIT License.* URL: <https://opensource.org/license/mit>.
- [24] INCIBE. *Cómo redactar un aviso legal.* URL: <https://www.incibe.es/protege-tu-empresa/blog/como-redactar-aviso-legal>.
- [25] Agencia Española de Protección de Datos. *Guía para el cumplimiento del deber de informar.* URL: <https://www.aepd.es/es/documento/guia-deber-informar.pdf>.
- [26] Agencia Española de Protección de Datos. *Modelo de registro de actividades de tratamiento.* URL: <https://www.aepd.es/es/documento/actuaciones-tratamiento-responsables.pdf>.

- [27] BOE. *Real Decreto 311/2022, Esquema Nacional de Seguridad*. URL: <https://www.boe.es/buscar/act.php?id=BOE-A-2022-14698>.
- [28] Grupo Vadillo Asesores. *¿Qué aspectos legales debe cumplir una página web?* URL: <https://www.grupovadillo.com/aspectos-legales-pagina-web/>.
- [29] Tailwind Labs Inc. *Tailwind CSS*. URL: <https://tailwindcss.com/>.
- [30] WorkOS. *Radix UI*. URL: <https://www.radix-ui.com/>.
- [31] Tauri. *Tauri Framework*. URL: <https://tauri.app/>.