
This is the **published version** of the bachelor thesis:

Pedro Pascal, Pol. *Disseny i implementació d'un assistent d'inventari domèstic intel·ligent basat en Processament de Llenguatge Natural (PLN) i Raonament Semàntic*. Treball de Final de Grau (Universitat Autònoma de Barcelona), 2026 (Enginyeria Informàtica)

This version is available at <https://ddd.uab.cat/record/334473>

under the terms of the  license.

Sistema d'Assistència a la Vida Diària per a la Gestió Intel·ligent d'Objectes Domèstics mitjançant Processament de Llenguatge Natural

Pol Pedro Pascal

Resum-La dificultat per localitzar objectes dins l'habitatge és un problema quotidià que afecta especialment persones grans o amb deteriorament cognitiu i llars amb múltiples integrants. Aquest treball presenta el disseny, la implementació i l'avaluació d'una aplicació mòbil d'assistència a la vida diària (*Ambient Assisted Living*, AAL) que permet gestionar l'inventari domèstic mitjançant llenguatge natural per veu i text, sense sensors ni dispositius addicionals, utilitzant únicament el telèfon mòbil. El sistema s'articula sobre una arquitectura *Retrieval-Augmented Generation* (RAG) real, que combina la recuperació híbrida sobre Azure AI Search amb un model generatiu (gpt-4.1), i sobre un model multi-tenant amb llars compartides entre diversos usuaris. Els resultats mostren un sistema funcional d'extrem a extrem, amb el nucli de llenguatge natural verificat amb dades reals.

Paraules clau-Ambient Assisted Living, llenguatge natural, models de llenguatge (LLM), Retrieval-Augmented Generation, inventari domèstic, cerca semàntica, accessibilitat.

Abstract-Finding objects at home is a daily problem that especially affects older people, individuals with cognitive impairment and households with several members. This work presents the design, implementation and evaluation of a mobile Ambient Assisted Living (AAL) application that lets users manage a household inventory through natural language, by voice and text, without sensors or additional devices, using only the smartphone. The system is built on a real Retrieval-Augmented Generation (RAG) architecture, combining hybrid retrieval over Azure AI Search with a generative model (gpt-4.1), and on a multi-tenant model with homes shared among several users. The results show an end-to-end functional system, with the natural-language core verified against real data.

Index Terms-Ambient Assisted Living, natural language, large language models (LLM), Retrieval-Augmented Generation, household inventory, semantic search, accessibility.



1 INTRODUCCIÓ I CONTEXTUALITZACIÓ

La dificultat per localitzar objectes dins l'habitatge és un problema quotidià que provoca pèrdues de temps, malbaratament de recursos i situacions d'estrès. Aquesta problemàtica afecta gran part de la població, però especialment a persones amb dificultats de memòria o deteriorament cognitiu, persones grans o simplement llars amb múltiples integrants on els objectes es mouen freqüentment.

En aquest context, els sistemes d'*Ambient Assisted Living* (AAL) [3] representen un camp de desenvolupament tecnològic que es centra en facilitar la vida diària de les persones mitjançant solucions digitals que augmenten l'autonomia i el benestar dins l'entorn domèstic.

En els últims anys, l'exponencial creixement i integració de models d'intel·ligència artificial (IA), especialment aquells basats en processament de llenguatge natural (LLMs) [5], han ampliat el potencial i les possibilitats d'aquests siste-

mes. Aquestes tecnologies permeten crear solucions més accessibles i naturals per a l'usuari, basades en la interacció mitjançant el llenguatge natural. D'aquesta manera, els sistemes AAL poden potenciar-se amb l'ús d'aquests models d'IA, permetent gestionar informació sobre la llar sense la necessitat de portar o d'instal·lar dispositius addicionals, utilitzant el telèfon mòbil com única eina i interfície d'interacció.

A partir d'aquesta premissa, el present treball proposa una aplicació mòbil que permet registrar, consultar, moure i eliminar objectes de la llar simplement parlant o escrivint, delegant en un model de llenguatge la interpretació de les instruccions. La resta del document s'organitza de la manera següent: la secció 2 fixa els objectius; la secció 3 situa el treball respecte a l'estat de l'art; la secció 4 descriu la metodologia de treball i l'arquitectura del sistema; la secció 5 presenta els resultats funcionals assolits i hi inclou un enllaç a un vídeo demostratiu que recorre tota l'aplicació; i la secció 6 recull les conclusions i les línies de continuació. Es completa amb una reflexió sobre l'ús de la IA generativa, els agraïments i la bibliografia.

- E-mail de contacte: polpedropascal1@gmail.com
- Menció realitzada: Computació
- Treball tutoritzat per: David Gutiérrez Yana
- Curs 2025/26

2 OBJECTIUS

L'objectiu principal era dissenyar, implementar i avaluar una aplicació mòbil d'assistència a la vida diària que permetés la gestió intel·ligent de l'inventari domèstic mitjançant llenguatge natural. Per aconseguir-ho, s'han utilitzat models de llenguatge (LLM) integrats mitjançant una arquitectura *Retrieval-Augmented Generation* (RAG), que interpreten les consultes de l'usuari expressades amb llenguatge natural per veu o text. Aquesta arquitectura ha permès combinar la capacitat d'integració dels models LLMs amb informació estructurada emmagatzemada en la base de dades del sistema.

A partir d'aquest objectiu general, es van definir sis objectius específics:

1. **Anàlisi i model de dades.** Dissenyar un model de dades jeràrquic (usuari → habitatge → habitació → contenidor → objecte) capaç de representar l'inventari domèstic i les relacions de pertinença entre els seus elements.
2. **Disseny arquitectònic.** Definir una arquitectura Frontend-Backend-LLM-Base de dades i el flux RAG que la recorre, garantint la separació de responsabilitats i la integració dels serveis al núvol.
3. **Backend i persistència.** Implementar un backend complet amb les operacions CRUD sobre l'inventari, l'autenticació d'usuaris i el model multi-tenant de llars compartides.
4. **Integració amb el LLM.** Integrar un model de llenguatge per a la interpretació de la intenció i l'extracció d'entitats, amb sortides estructurades i desambiguació semàntica quan la confiança és baixa.
5. **Aplicació mòbil.** Desenvolupar el client mòbil amb entrada per veu i text, la gestió de comptes, llars i invitacions, i una vessant d'accessibilitat orientada a l'usuari AAL.
6. **Validació i avaluació.** Validar el sistema mitjançant proves funcionals i escenaris reals d'ús, incloent-hi llars amb múltiples usuaris.

A més a més, dins l'objectiu 3 s'incloïa la implementació d'un model multi-tenant en què cada compte pogués gestionar diverses llars de manera independent. Dins de cada llar, l'usuari pot convidar altres usuaris, permetent que tots ells puguin consultar, afegir o modificar la ubicació dels objectes registrats. Aquesta funcionalitat permet que la informació es mantingui compartida i actualitzada per a tots els membres de la llar, facilitant la gestió conjunta dels objectes i la seva localització.

3 ESTAT DE L'ART

Diversos treballs dins l'àmbit dels sistemes AAL han explorat solucions per ajudar les persones a gestionar activitats quotidianes dins la llar. Moltes d'aquestes propostes es basen en l'ús de sensors, càmeres o dispositius portables que permeten monitoritzar l'entorn domèstic i detectar activitats o la ubicació d'objectes. Aquestes solucions sovint combinen tecnologies com la visió per computador, sensors ambientals o dispositius IoT per proporcionar assistència contextual a l'usuari i millorar la seva autonomia dins l'habitatge.

En l'àmbit de la recerca per trobar solucions amb enfocaments o propòsits similars, s'han trobat propostes com

MemPal [2], que planteja un assistent de memòria portable equipat amb una càmera capaç d'analitzar automàticament on l'usuari va desant diferents objectes dins la llar. El sistema registra aquesta informació i permet posteriorment realitzar consultes de veu per recuperar-la. Aquest enfocament, però, està enfocat únicament a donar assistència a persones grans o amb demència, amb la utilització d'un dispositiu addicional que l'usuari ha de portar constantment, mentre que el projecte proposat en aquest treball busca evitar aquesta dependència de sensors o dispositius externs i basar-se únicament en la interacció amb el telèfon mòbil.

Paral·lelament, també existeixen solucions orientades a la gestió d'inventaris domèstics. Algunes aplicacions comercials permeten registrar la ubicació d'objectes dins la llar i consultar-ho posteriorment, facilitant l'organització i gestió de l'emmagatzematge. Un exemple és *HomeBox* [1], una aplicació que permet catalogar objectes i registrar-ne la ubicació dins l'habitatge, així com compartir aquesta informació entre diversos usuaris. Tot i això, aquest tipus d'eines estan orientades principalment a la catalogació d'objectes o a la gestió d'inventaris estàtics, i acostumen a basar-se en interfícies tradicionals d'escriptori o en cerques textuals.

En contrast amb aquestes solucions, el projecte proposat se centra en facilitar la gestió dinàmica d'objectes dins la llar i en proporcionar una interacció més natural amb el sistema mitjançant veu o text. A més, el sistema està pensat per ser utilitzat en llars amb diversos membres, permetent compartir la informació sobre la ubicació dels objectes sense necessitat d'instal·lar sensors o dispositius addicionals ni de portar cap equipament específic. Aquesta aproximació busca reduir la càrrega tecnològica per a l'usuari i facilitar la integració del sistema en el dia a dia. El diferencial respecte a l'estat de l'art queda, doncs, clar: ni la dependència d'un dispositiu portable de *MemPal* ni la naturalesa estàtica i d'escriptori de *HomeBox*, sinó una gestió dinàmica, compartida i conversacional únicament amb el telèfon mòbil.

4 METODOLOGIA

4.1 Procés de desenvolupament

Per assolir els objectius específics en el termini de cinc mesos, s'ha emprat una metodologia àgil basada en Scrum [6], seguint el cicle de sprints de dues setmanes. En cada sprint es comença amb la planificació (definició i estimació) de les tasques a desenvolupar. Durant l'sprint es realitzen les tasques de forma iterativa i, un cop aquest finalitza, es fa retrospectiva del treball dut a terme i es planifica el següent sprint. Aquesta metodologia és molt recomanada per a projectes de desenvolupament de programari amb components experimentals, com és el cas de la integració de models de llenguatge (LLMs).

A més a més, aquesta elecció es justifica, en primer lloc, per la naturalesa iterativa i incremental de Scrum, que permet fer i lliurar versions funcionals del sistema al final de cada sprint, facilitant la detecció anticipada d'errors i la validació contínua dels requisits. Aquest és un aspecte especialment rellevant per als objectius 3, 4 i 5, ja que presenten

dependències, riscos tècnics i són susceptibles a possibles canvis o ajustos durant el desenvolupament. En segon lloc, la flexibilitat de Scrum permet fer canvis en els requisits entre sprints i ajustar desviacions en l'estimació del temps de les tasques. Aquesta característica ha resultat especialment útil durant el desenvolupament en aspectes crítics com la selecció i la calibració del model de LLM, que ha requerit modificacions i noves implementacions en funció dels resultats obtinguts en les primeres iteracions.

4.2 Arquitectura del sistema

El sistema s'estructura en quatre capes: un client mòbil (Frontend), una API (Backend), un model de llenguatge (LLM) i una capa de dades formada per una base de dades relacional i un motor de cerca semàntica. El Frontend s'ha desenvolupat amb Flutter [14]; el Backend, amb FastAPI; el model generatiu i els *embeddings* s'obtenen d'Azure OpenAI; i la persistència es reparteix entre PostgreSQL, que emmagatzema l'inventari real i les seves relacions estructurals, i Azure AI Search, que indexa una representació de-normalitzada de cada objecte per a la recuperació semàntica.

Pel que fa al desplegament a Azure, l'arquitectura es reparteix en dos **tenants** diferenciats. El primer és el *tenant* de recursos, que conté tota la infraestructura del sistema -el grup de recursos amb Container Apps, el registre de contenidors, PostgreSQL, AI Search, Azure OpenAI, Speech, Key Vault i Application Insights-. El segon és un *tenant* independent de Microsoft Entra External ID (CIAM), dedicat exclusivament a custodiar les identitats dels usuaris finals. Aquesta separació segueix el patró habitual de *Customer Identity and Access Management*: les identitats dels usuaris viuen aïllades en el seu propi *tenant*, desacobrades de la infraestructura, de manera que la gestió de comptes (registre, inici de sessió, rotació de claus) queda delegada al servei d'identitat i no es barreja amb els recursos d'aplicació.

La Fig.1 il·lustra l'arquitectura de desplegament completa del sistema sobre Azure, mostrant la distribució dels components entre els dos *tenants* diferenciats. Per llegibilitat s'han omès alguns recursos de suport no essencials per al flux principal: la xarxa virtual (aal_llm_vnet) i el NSG integrats a l'entorn de Container Apps, el Log Analytics Workspace (que rep els logs de l'entorn i d'Application Insights) i el Storage Account.

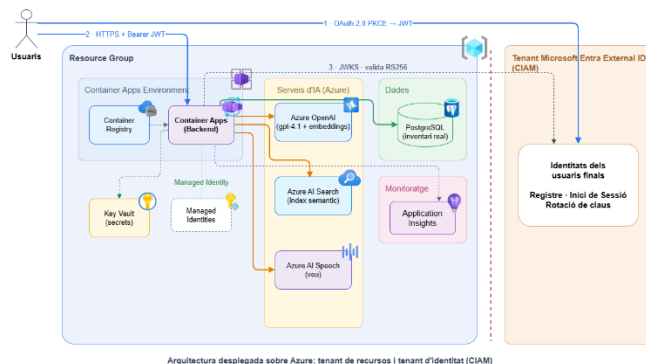


Fig.1. Arquitectura de desplegament del sistema sobre Azure amb els dos *tenants* diferenciats.

El diagrama mostra dos blocs principals separats. El *tenant* de recursos agrupa tota la infraestructura d'aplicació: Container Apps (on s'executa el Backend en contenidors Docker), Container Registry (que emmagatzema les imatges), PostgreSQL Flexible Server (la base de dades relacional), Azure AI Search (el motor de cerca híbrida), Azure OpenAI (el model generatiu i els *embeddings*), Azure Speech (el reconeixement i la síntesi de veu), Key Vault (la gestió centralitzada de secrets) i Application Insights (la telemetria). El *tenant* d'Entra External ID (CIAM), independent del primer, conté exclusivament les identitats dels usuaris finals. Aquesta separació segueix el patró CIAM estàndard: la gestió de comptes queda delegada al servei d'identitat sense barrejar-se amb els recursos d'aplicació. El Backend valida els tokens JWT emesos per aquest *tenant* en cada petició, garantint l'autenticació sense emmagatzemar mai contrasenyes ni sessions pròpies.

4.3 Flux de processament (RAG)

Abans de descriure el flux convé justificar per què s'articula com una arquitectura RAG i no com una simple crida a un model de llenguatge. Un LLM, per si sol, és capaç d'interpretar una frase, però desconeix l'inventari real de cada llar: no sap quins objectes hi existeixen, ni on són, ni si n'hi ha de noms semblants. Si se li delegués la resolució completa de l'operació, tendiria a inventar ubicacions o a donar per bons objectes inexistents. D'aquí l'elecció d'una arquitectura RAG [4]: el model aporta exclusivament allò en què és fort -la comprensió del llenguatge i la generació de la resposta-, mentre que la **identitat i la ubicació dels objectes s'ancoren sempre en dades reals** recuperades de la base de dades i de l'index semàntic. Enfront d'un *function calling* directe, en què el model decidiria pel seu compte sobre quin objecte actua, recuperar primer mitjançant la cerca híbrida permet resoldre l'ambigüitat de noms -"martell" o "Martell", objectes semblants en calaixos diferents- amb un senyal de rellevància objectiu i demanar aclaració quan cal, abans d'executar cap canvi irreversible sobre l'inventari.

El flux comença quan l'usuari introdueix una instrucció a l'aplicació mòbil per text o veu (per exemple, "He deixat les claus al calaix de l'entrada" o "On són les ulleres?"). El Backend envia el text al LLM, que en detecta la intenció (insert, query, move, remove, create_room, create_container, clarify o history), extreu les entitats rellevants (objecte, ubicació, contenidor, descripció) i les normalitza a l'idioma canònic de la llar, produint una estructura JSON consistent. Amb aquesta representació, el Backend resol la identitat de l'objecte mitjançant la cerca híbrida sobre Azure AI Search; segons el resultat, executa l'operació corresponent sobre PostgreSQL, demana una aclaració a l'usuari si hi ha ambigüitat, o el tracta com a objecte nou. Finalment, el Backend torna a delegar en el LLM per transformar el resultat estructurat en una resposta en llenguatge natural, en l'idioma de l'usuari, que s'envia de tornada a l'aplicació. Cada canvi rellevant sobre l'inventari re-sincronitza el document corresponent a l'índex semàntic, de manera que la cerca es manté coherent amb l'estat real de la base de dades.

A tall d'exemple, davant la instrucció "He deixat les claus al

calaix de l'entrada" el model produeix una estructura normalitzada com la següent, que és la representació amb què el Backend opera la resta del flux:

```
{
  "intent": "insert",
  "object": "claus",
  "description": "joc de claus",
  "location": { "room": "entrada", "container": "calaix" }
}
```

Convé matisar l'abast del terme RAG en aquest context. En les consultes -"on són les ulleres?" (intent *query*) o les preguntes d'historial (*history*)- el flux és RAG en sentit estricte: es recupera informació real de l'inventari i el model la verbalitza. En les ordres que muten l'inventari (*insert*, *move*, *remove*), en canvi, la recuperació no alimenta la generació de la resposta sinó que resol la **identitat de l'objecte** sobre el qual s'actuarà, de manera que el patró s'acosta més al *tool use*. Tot i aquesta diferència de matís, el treball es refereix al flux de manera unificada com a RAG perquè tots els camins comparteixen la mateixa columna vertebral -interpretació amb el LLM, ancoratge en dades reals mitjançant la cerca i resposta en llenguatge natural-, que és justament el que distingeix el sistema d'una simple crida generativa.

La Fig.2 il·lustra els dos carrils principals del flux de processament RAG, que comparteixen la mateixa arquitectura però divergeixen en com s'utilitza el resultat de la recuperació.

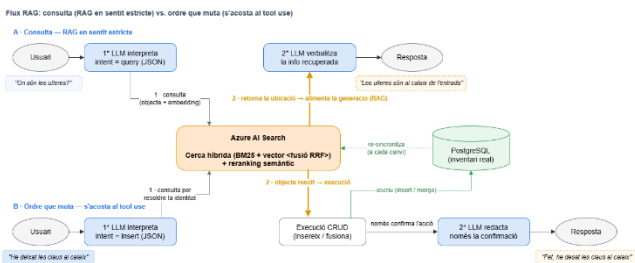


Fig.2. Flux de processament RAG amb els dos carrils d'operació: consulta (A) i ordre que muta l'inventari (B). Azure AI Search és una peça compartida pels dos carrils. BM25 i la cerca vectorial conformen la cerca híbrida (fusió per RRF); el *reranker* semàntic hi afegeix una capa addicional d'ordenació. El carril A (consulta, intents *query* o *history*) és RAG en sentit estricte: el document recuperat de l'índex -que conté la ubicació de l'objecte, sincronitzada des de PostgreSQL- alimenta directament la generació del LLM. El carril B (ordres que muten, intents *insert*, *move* o *remove*) mostra la diferència clau: la mateixa cerca resol la identitat de l'objecte sobre el qual actuar; un cop identificat, s'escriu a PostgreSQL, l'índex es re-sincronitza, i el LLM únicament confirma l'acció en llenguatge natural, sense que el contingut recuperat alimenti la generació. Aquesta via s'acosta més al *tool use*. Tot i la diferència, ambdós carrils comparteixen la columna vertebral: interpretació LLM, ancoratge en dades reals via cerca i resposta en llenguatge natural.

4.4 Desenvolupament incremental i justificació de

l'stack

El desenvolupament del sistema s'ha organitzat en set fases incrementals que encaixen amb el cicle d'sprints descrit al punt 4.1, de manera que cada fase deixava una part del sistema funcional i verificable abans de passar a la següent. Les cinc primeres fases construeixen el Backend de baix a dalt: la fase 1 estableix l'estructura base del projecte, la càrrega de secrets i la connexió asíncrona a la base de dades; la fase 2 incorpora l'autenticació amb Microsoft Entra External ID; la fase 3, la gestió de llars i el sistema d'invitacions; la fase 4, les ubicacions (habitacions i contenidors); i la fase 5, els objectes i la integració amb Azure AI Search. Les dues darreres fases afegeixen la capa intel·ligent i la d'interacció: la fase 6 implementa el *pipeline* de llenguatge natural amb el LLM i la fase 7, la integració de veu. Aquest ordre no és arbitrari, sinó que respon a les dependències del sistema -no es pot interpretar una instrucció sobre objectes sense tenir abans el model de dades, la cerca i l'autenticació-, i és coherent amb la prioritització de riscos tècnics que justifica l'ús de Scrum.

Pel que fa a les decisions tecnològiques, l'elecció de l'*stack* -FastAPI [7], PostgreSQL amb SQLAlchemy 2.0 [8], Pydantic [9] i Azure Container Apps- es justifica per la naturalesa asíncrona i orientada a E/S del Backend; la justificació detallada de cadascuna es descriu a l'Apèndix 1.

5 RESULTATS

A continuació es descriu el que s'ha implementat i verificat realment, organitzat en les set àrees funcionals del sistema i tancat amb una secció transversal sobre el model de seguretat. Com a complement a aquesta descripció, s'adjunta un enllaç a un **vídeo explicatiu** en què es navega per tota l'aplicació mostrant-ne les funcionalitats: <https://youtu.be/skJctBXZtk?feature=shared>.

5.1 Interacció per llenguatge natural

5.1.1 El pipeline i el contracte estructurat

És el nucli diferenciador del treball: gestionar l'inventari parlant o escrivint, sense formularis. El *pipeline* de processament del llenguatge natural interpreta la intenció amb el LLM (gpt-4.1, sortida JSON estricta, *temperature* 0,1), executa l'operació CRUD corresponent delegant la localització de l'objecte en la cerca semàntica, i genera la resposta natural amb el mateix model (*temperature* 0,5) en l'idioma de l'usuari. En interpretar la frase, el model **normalitza les entitats extretes a l'idioma canònic de la llar** (vegeu l'apartat 5.5), de manera que els noms d'objectes, habitacions i contenidors s'emmagatzemen i s'indexen sempre en un únic idioma i la cerca BM25 es manté consistent; en generar la resposta, en canvi, el sistema tradueix implícitament aquests noms a l'idioma preferit de l'usuari. La conversa és multi-torn però *stateless*: el client reenvia els darrers torns de context en cada petició.

Aquesta interpretació es recolza en una decisió d'implementació central: el model no respon en text lliure, sinó mitjançant *Structured Outputs* [13] amb un esquema JSON estricta (*strict: true*). Així, OpenAI garanteix a nivell d'API que la sortida compleix exactament l'esquema -sense ometre camps, afegir-ne d'extra ni equivocar els tipus-,

cosa que elimina els reintents per format i la validació defensiva. L'esquema defineix un contracte de camps que el model ha d'emplenar a cada interpretació. El detall complet dels camps de l'esquema -`from_location`, `quantity`, `confidence`, `needs_clarification`, `clarification_options` i `message`- i la seva semàntica es descriuen a l'Apèndix 2.

Sobre aquest contracte s'articula la resta del *pipeline*. La interpretació i la redacció de la resposta es resolen amb dues crides separades al LLM -una amb `temperature 0,1` i esquema estricte per encertar l'estructura, i una altra amb `temperature 0,5` i text lliure per sonar natural-, i si la segona falla s'utilitza el camp `message` de la primera com a *fallback*, de manera que l'usuari sempre rep una resposta. Com que el model retorna els noms d'habitacions i contenidors en text lliure, dues funcions de resolució els converteixen en registres reals comparant sense distingir majúscules (`ilike`) i acotant per habitació quan cal desfer ambigüitats. Tot plegat s'orquestra en quatre passos -verificació d'accés, interpretació, execució segons la intenció i generació de la resposta- i la desambiguació opera en dos nivells complementaris: el de llenguatge que acabem de descriure (`needs_clarification`) i el de cerca, que decideix sobre el *reranker* tal com s'explica al punt 5.3.

5.1.2 El disseny dels prompts

Si l'esquema fixa *què* ha de retornar el model, és el **disseny dels prompts** el que determina *com* arriba a fer-ho, i és aquí on recau bona part de la robustesa del sistema. El nucli és un *system prompt* d'extracció d'intenció parametritzat per idioma que, més enllà de definir els vuit intents, recull una **bateria de regles numerades**. Cap d'aquestes regles és redacció lliure: cadascuna respon a un comportament erroni observat empíricament durant el desenvolupament. Així, la regla de **normalització l'idioma canònic** obliga el model a emetre tots els noms d'entitat en l'idioma de la llar “sigui quin sigui l'idioma del missatge” -contrarestant la seva tendència natural a fer-ho en la llengua de l'usuari- i és la peça de la qual depèn la consistència del BM25 (descriu a l'apartat 5.5) i la cerca interlingüe (descriu a l'apartat 5.3). La distinció explícita entre *history* i *query* evita que les preguntes en passat es colin com a consultes de la ubicació actual, un fals positiu perillós. La regla que fa **opcional l'origen d'un moviment** -“mai no demanis d'on surt l'objecte; un moviment sense origen no està incomplet”- va corregir un cas real en què l'assistent reclamava informació supèrflua, i és un exemple net de la **fragilitat** d'aquests sistemes: el comportament percebut depèn de la formulació d'una sola frase del prompt. En la mateixa línia, perquè una ubicació “sense habitació” es pogués tractar com una més, el prompt indueix el model a emetre un **literal sintètic** (`__unassigned__`) en lloc d'una frase natural, amb una funció de detecció en codi com a xarxa de seguretat.

Un episodi concret el lustra que el prompting és **disseny i no simple redacció**. La cerca interlingüe del cercador manual es va intentar resoldre inicialment reaprofitant el mateix classificador d'intenció; davant d'una paraula solta -el que s'escriu en una barra de cerca- el model la classificava de manera **no determinista** com a ambigua i de vegades no n'extreia l'objecte, de manera que el resultat depenia de

l'atzar. La solució no va ser un prompt més llarg sinó un prompt **mono-tasca i acotat**: un traductor dedicat (explicat al punt 5.3) amb una única instrucció -traduir el terme i retornar-lo, sense intenció ni missatge- que fa el comportament fiable. La mateixa eina de model, amb un prompt mal encaixat dona un resultat aleatori i, amb un prompt ajustat al problema, un de determinista.

Finalment, l'elecció de la **temperatura** és part del mateix disseny: 0,1 a la interpretació per prioritzar l'encert estructural, 0,5 a la generació de la resposta perquè soni natural, i 0,0 al traductor del cercador per garantir determinisme. En conjunt, aquests prompts es comporten com un **contracte viu**: diverses millores de robustesa del sistema no van ser canvis de codi sinó ajustos en la formulació de les instruccions, calibrats provant el comportament real del model.

5.1.3 Historial i desambiguació

Sobre aquest nucli s'han construït diversos desenvolupaments. El primer és la **consulta de moviments per llenguatge natural** (intent *history*), que respon preguntes temporals i d'historial sobre els objectes -què s'ha mogut, d'on a on, quan, qui ho ha fet i què ha canviat en una habitació-. A diferència de la resta d'intencions, *history* no consulta la cerca semàntica sinó la taula `object_movements` de PostgreSQL, amb àmbit per objecte, per habitació o global i una finestra temporal interpretada pel LLM; el resultat és una llista acotada de moviments que es verbalitzen en la resposta. D'aquesta manera, l'historial que ja registren els *triggers* de la base de dades es fa accessible simplement parlant. El segon és la **desambiguació amb mínim de rellevància**, que en localitzar un objecte únic classifica la cerca en *no trobat*, *aclaració* o *coincidència* sobre el *reranker* semàntic d'Azure AI Search per evitar tant els falsos positius d'aclaració davant d'objectes inexistents com les confusions entre objectes semblants; el seu funcionament i la calibració empírica dels llindars es detallen al punt 5.3. A aquests s'hi afegeixen la normalització bilingüe d'entitats (apartat 5.5) i l'acció pendent per quantitat (apartat 5.2), descrites als seus apartats respectius.

5.1.4 L'assistent de veu i el mode conversacional

Pel que fa a l'**assistent de veu**, peça clau de la vessant AAL, s'ha implementat el reconeixement de veu per *streaming* sobre WebSocket amb Azure Speech en mode interactiu i detecció automàtica de fi de parla (VAD), amb *fallback* temporal i parada manual; la síntesi de veu (TTS) per locutar les respostes; una màquina de set estats (Idle, Listening, Processing, Responding, Clarifying, Confirming, Error) amb una barra d'assistent persistent; i un *token broker* de Speech perquè la clau no arribi mai al client.

La Fig.3 representa la màquina d'estats que governa el comportament de l'assistent de veu al llarg d'una sessió conversacional.

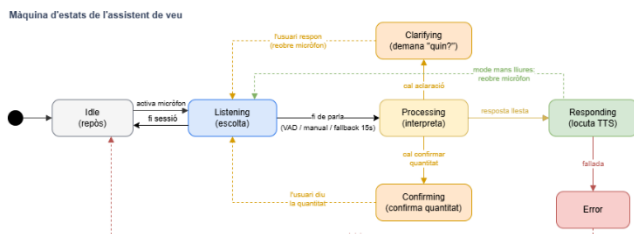


Fig.3. Màquina d'estats de l'assistent de veu amb els set estats i les transicions principals.

La màquina consta de set estats: Idle (en repòs, esperant que l'usuari iniciï la interacció), Listening (capturant l'àudio de l'usuari), Processing (processant la transcripció i consultant el Backend), Responding (locuant la resposta en TTS), Clarifying (esperant que l'usuari respongui a una pregunta de desambiguació), Confirming (esperant confirmació d'una acció) i Error (estat d'error recuperable). El tancament de l'escolta es produeix per tres mecanismes complementaris: el VAD del servei de reconeixement de veu, que detecta el final de la parla quan es fa el silenci; la parada manual per part de l'usuari; i un *fallback* temporal de 15 segons com a xarxa de seguretat. En el mode de mans lliures, en acabar la locució (estat Responding), el micròfon es reobre automàticament perquè l'usuari pugui respondre sense tocar la pantalla. Els estats Clarifying i Confirming retornen a Listening per permetre que l'usuari contesti parlant; si no es detecta cap parla en el temps d'espera, la sessió es finalitza de manera neta.

Els detalls del flux d'àudio PCM, els mecanismes de tancament i els marges temporals es descriuen a l'Apèndix 3.

Sobre aquest reconeixement s'ha construït un **mode conversacional de mans lliures** que encadena torns sense que l'usuari hagi de tocar la pantalla. Quan una resposta de l'assistent queda a l'espera de rèplica -una aclaració, la confirmació d'una quantitat o una resposta oberta-, el sistema en locuta el text amb TTS i, en acabar la reproducció, **reobre automàticament el micròfon** perquè l'usuari contesti parlant; si l'usuari prefereix tocar el micròfon mentre la veu encara sona, la locució s'interromp a l'instant i comença l'escolta. Les respostes parlades es resolen localment abans de tornar a recórrer al model: un "sí" o un "no" encaminen la continuació o el tancament de la conversa, i una quantitat dita en veu alta ("dues", "totes") resol l'acció pendent per unitats (descriu al punt 5.2) sense una nova crida al LLM. Si el VAD o el temps d'espera no capten cap parla, el sistema no tanca la conversa sinó que **restaura l'estat d'espera** mantenint les targetes visibles, de manera que l'usuari pot tornar a intentar-ho. Aquest comportament és especialment rellevant en el Mode Senior (explicat a l'apartat 5.6), on l'objectiu és minimitzar la interacció tàctil.

5.1.5 Continuitat del diàleg i coherència de la interfície

Tot i que la conversa amb el Backend és *stateless*, el client manté la continuïtat del diàleg reenviant una finestra dels últims cinc intercanvis en cada petició, i invalida selectivament la memòria cau de les pantalles afectades després de cada acció que muta l'inventari. La mecànica detallada d'aquesta gestió es descriu a l'Apèndix 4.

5.2 Gestió de l'inventari (model jeràrquic CRUD)

Sobre l'estructura Llar → Habitació → Contenidor (recursiu) → Objecte s'ha implementat el CRUD complet de tots els elements, inclosos els àlies d'objecte i l'arbre de contenidors niats. Com a invariant de domini, tot objecte té habitació i el contenidor és opcional. Els *triggers* de PostgreSQL registren cada inserció, moviment i eliminació, que es mostren al frontend amb temps relatiu. El reanomenament està restringit per disseny: només els objectes es reanomenen i re-sincronitzen l'índex, mentre que habitacions i contenidors només canvien icona o tipus. Més enllà d'aquest nucli CRUD, la gestió cobreix dos casos no trivials que mereixen un desenvolupament a banda -la quantitat d'unitats d'un mateix objecte i l'esborrat d'una habitació-, que es detallen més avall. El model de dades relacional complet -taules, tipus enumerats, *triggers*, vistes i mecanisme de sincronització amb l'índex semàntic- es descriu a l'Apèndix 5.

Pràcticament tots els objectes d'una llar són únics, però alguns existeixen en diverses unitats idèntiques (gots, piles, bombetes). El sistema ho modela amb un **comptador 'quantity' per fila**, no amb una fila per unitat: cinc gots iguals són una sola fila amb quantitat cinc, sense identitat individual de cada unitat. La quantitat per defecte és 1 i una restricció de base de dades garanteix que mai sigui nul·la o negativa -"zero unitats" equival a no existir i es modela esborrant la fila-. Sobre aquest model s'implementa moure i eliminar **per unitats**, amb la lògica centralitzada en un únic mòdul del qual depenen tant l'API REST com el *pipeline* NLP. Moure una part de les unitats provoca una **partició**: es decremента l'origen i, al destí, o bé es **fusi**ona amb una fila idèntica ja existent -mateix nom (insensible a majúscules) i mateixa ubicació exacta- o bé se'n crea una de nova heretant els àlies. El mateix criteri de fusió s'aplica quan l'assistent **insereix** un objecte que ja existeix al mateix lloc: en comptes de duplicar la fila n'incrementa la quantitat, de manera que dir "*afegeix un martell al traster*" on ja n'hi ha un suma les unitats sense que una diferència de grafia ("Martell" / "martell") les fragmenti. Eliminar una part només decremента el comptador, i eliminar-les totes esborra la fila i el seu document de l'índex. Quan una ordre de veu no especifica quantes unitats afectar i n'hi ha més d'una, el sistema no executa res: retorna una **acció pendent** perquè l'usuari esculli la quantitat (un número o "totes") abans de confirmar. A la interfície, un distintiu numèric apareix només quan hi ha més d'una unitat, i el selector de quantitat només es mostra quan té sentit. Cal reconèixer-ne una limitació actual: l'historial de moviments no enregistra quantitats, de manera que els canvis parcials de quantitat no hi queden reflectits; és una millora natural per a treball futur.

Quan s'esborra una habitació, els contenidors que conté s'eliminen sempre en cascada, mentre que pels objectes el sistema ofereix dues estratègies que l'usuari tria a l'aplicació: **eliminar-los** -cosa que també neteja els seus documents de l'índex semàntic- o **desassignar-los**, que els deixa en un estat *pendent* (sense habitació ni contenidor) tot registrant el moviment i re-sincronitzant l'índex perquè la cerca reflecteixi que han perdut la ubicació. Aquesta segona via materialitza l'únic estat en què un objecte pot exis-

tir sense habitació: a la pantalla d'inici apareix un avís dels objectes pendents que porta a una vista dedicada per reassignar-los, moment en què tornen a indexar-se i a aparèixer a les cerques. La mecànica de com l'assistent de veu accedeix i gestiona aquests objectes pendents, a diferència del cercador manual, es detalla a l'Apèndix 5.

5.3 Cerca híbrida semàntica

5.3.1 El motor i la sincronització

El motor de recuperació alimenta tant la cerca manual com el *pipeline* NLP. Combina BM25, cerca vectorial (HNSW [17], distància cosinus, 1536 dimensions) i *reranking* semàntic sobre Azure AI Search, amb *embeddings* generats amb *text-embedding-3-small* i un text d'indexació que incorpora la ubicació de l'objecte. La sincronització de PostgreSQL cap a l'índex es fa amb un document denormalitzat per objecte que es re-sincronitza en cada canvi rellevant, amb neteja explícita de l'índex en esborrar una llar o un objecte.

5.3.2 Puntuació i regla de decisió

Pel que fa al funcionament intern de la cerca, convé detallar com es puntuen i s'ordenen els resultats, perquè el sistema treballa amb **dues puntuacions de naturalesa diferent**. En primer lloc, la cerca recupera els *top* deu candidats (filtrats per llar i exclouent-ne els objectes pendents d'assignar) combinant els dos senyals de recuperació -BM25 [15] i vector- mitjançant *Reciprocal Rank Fusion* (RRF) [10], [16]: la fusió no és una suma lineal ponderada dels dos *scores*, sinó que combina la **posició** de cada document en cadascun dels dos rànquings, i en resulta el *score* híbrid (@search.score). En segon lloc, sobre aquests candidats actua el *reranker* semàntic [11], [18], que els **torna a ordenar avaluant la rellevància semàntica de nou, amb independència del *score* híbrid previ**, i assigna a cadascun una puntuació pròpia en una escala de 0 a 4 (@search.rerankerScore). És sobre aquesta segona puntuació, la del *reranker*, que opera la regla de decisió: si el millor resultat no supera un **mínim absolut de rellevància** (fixat a 1,8), es considera que l'objecte no existeix i s'evita el fals positiu de demanar aclaració; si el supera, s'avalua el *gap* relatiu* entre els dos primers candidats -la diferència normalitzada $(s_1 - s_2) / s_1$ -, si és inferior a 0,1, els resultats es consideren massa semblants i el sistema demana a l'usuari que esculli. Tots dos llindars es van fixar empíricament mitjançant un banc d'avaluació propi, que mesura hit@1, hit@3 i MRR per al retrieval, i precisió i recall de la detecció d'aclariment, a més de la latència. El llindar mínim de rellevància (1,8) se situa per sobre del rang habitual dels objectes inexistents -que típicament no superen 1,73-, tot i que un cas extrem arriba fins a 1,96. Aquest cas excepcional s'explica per la naturalesa semàntica del *reranker*: la puntuació reflecteix la similitud entre la consulta i els objectes existents al catàleg, no si l'objecte existeix pròpiament. Quan la consulta descriu alguna cosa genèrica o propera semànticament a objectes reals -per exemple, un terme que comparteix característiques amb diversos objectes registrats-, el *reranker* pot assignar una puntuació alta tot i que l'objecte exacte no estigui inventariat. Aquest comportament és una limitació inherent de la cerca semàntica: similitud no im-

plica existència. Tot i això, en el context d'un inventari domèstic, aquest fals positiu és preferible a un fals negatiu: respondre "no existeix" quan l'objecte sí que hi és erosiona la confiança de l'usuari en el sistema, mentre que una pregunta de clarificació innecessària -"quin vols dir?"- té un cost molt menor i no compromet la utilitat del sistema. El *gap* presenta una meseta òptima entre 0,10 i 0,13. Amb aquests paràmetres calibrats, el retrieval assoleix el 100 % d'encert en el primer resultat (hit@1). El llindar mínim elimina també els falsos positius d'aclariment: sense aquest filtre, un objecte inexistent podia obtenir un *score* prou alt per superar el *gap* i provocar que el sistema demanés "quin?" en lloc de respondre que no existia; amb el llindar, aquests objectes queden descartats abans d'arribar a l'etapa d'aclariment.

La Fig.4 sintetitza el funcionament intern del motor de cerca, des de la consulta inicial fins a la decisió final sobre l'objecte identificat.

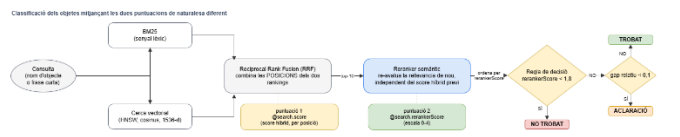


Fig.4. Pipeline de cerca híbrida i semàntica amb la regla de decisió del *reranker* (mínim 1,8 i *gap* 0,1).

La cerca híbrida fusiona BM25 i la cerca vectorial mitjançant *Reciprocal Rank Fusion* (RRF), que combina les posicions dels documents en cadascun dels dos rànquings -no els *scores* directament- i obté els deu candidats principals filtrats per llar. Sobre aquests candidats actua el *reranker* semàntic d'Azure AI Search, que els torna a ordenar avaluant la rellevància semàntica de nou, amb independència del *score* híbrid previ, i assigna a cadascun una puntuació pròpia en una escala de 0 a 4 (*rerankerScore*). A partir d'aquesta segona puntuació opera la regla de decisió en dos passos: primer, si el millor resultat no supera el mínim absolut de 1,8, l'objecte es considera inexistent i s'evita el fals positiu de demanar aclaració; si el supera, s'avalua el *gap* relatiu entre els dos primers candidats -la diferència normalitzada $(s_1 - s_2) / s_1$ -, si és inferior a 0,1, el sistema demana a l'usuari que esculli. Si supera el *gap*, l'operació s'executa directament sobre l'objecte identificat de manera unívoca.

La distribució empírica de les puntuacions sobre el banc d'avaluació i la justificació del sostre efectiu de l'escala es detallen a l'Apèndix 6. L'indicador de confiança que exposa aquest valor al cercador manual es descriu a l'Apèndix 7.

5.3.3 Cerca interlingüe

El cercador manual incorpora també una **cerca interlingüe**: abans de consultar l'índex normalitza el text introduït a l'idioma canònic de la llar (punt 5.5), de manera que escriure "spoon" en una llar configurada en castellà localitza igualment la "cuchara". Aquesta normalització no reaprofitja la interpretació d'intenció de l'assistent, sinó que es resol amb un **traductor dedicat i determinista** -una crida específica al model amb *temperature* 0,0 l'única tasca de la qual és traduir el terme i retornar-lo, sense intenció ni mis-

satge-. La raó és pràctica: el classificador d'intenció està pensat per a frases i, davant d'una paraula solta com la que s'escriu en un cercador, de vegades la tractava com a ambigua sense arribar a extreure'n l'objecte, de manera que el resultat interlingüe depenia de l'atzar del model; un traductor dedicat el fa fiable. Si la crida falla, el sistema recau de manera segura sobre el text original. A diferència del camí de l'assistent, el cercador **no aplica el mínim de rellevància**: retorna sempre els millors resultats ordenats per la semàntica i deixa que un percentatge de confiança baix comuniqui la feblesa d'una cerca sense coincidència real, en lloc d'amagar resultats.

5.4 Multi-tenancy: llars, membres i invitacions

El sistema permet que diverses persones -per exemple, cuidador i persona assistida, un cas típic d'AAL- comparteixin una llar. Cada compte pot pertànyer a diverses llars, amb verificació de membresia en cada operació. El flux d'invitacions és complet, amb estats (pendent, acceptada, rebutjada, caducada) i caducitat de set dies, i permet llistar les enviades vigents i cancel·lar-les. La gestió de membresia inclou abandonar la llar, expulsar membres i transferir-ne la propietat, amb esborrat en cascada de la llar i del seu índex quan en surt l'últim membre. Un usuari sense llar pot, en el primer accés, acceptar una invitació pendent sense haver-ne de crear cap.

El cas del propietari mereix una menció a banda. Un membre normal pot abandonar la llar lliurement, però un propietari amb altres membres ha de **transferir primer la propietat** a un successor abans de poder marxar -l'aplicació l'hi demana amb un selector de membre-; en canvi, si el propietari és l'únic integrant, abandonar equival a **esborrar la llar sencera** en cascada, inclosos els seus documents de l'índex. L'expulsió de membres, per la seva banda, queda reservada al propietari. Aquesta lògica garanteix que cap llar quedi mai sense propietari ni amb dades òrfenes a l'índex semàntic.

5.5 Autenticació i identitat

L'autenticació es basa en Microsoft Entra External ID (CIAM) [12] amb OAuth 2.0 PKCE [19]. Com s'ha avançat a la secció 4.2, les identitats viuen en un *tenant* d'Entra External ID independent del *tenant* de recursos, de manera que tota la gestió de comptes queda delegada al servei d'identitat i el Backend no emmagatzema mai contrasenyes ni sessions.

La validació dels tokens al Backend segueix el flux estàndard d'OpenID Connect. En rebre una petició, el servidor extreu el JWT de la capçalera *Authorization* i, a partir del punt de descobriment OIDC del *tenant*, obté la URL de les claus públiques (JWKS) i l'emissor esperat. Descarrega les claus RSA i les manté en memòria cau per no repetir la petició a cada crida; quan arriba un token signat amb una clau que no té a la memòria cau -cas típic de la **rotació periòdica de claus** de Microsoft Entra-, invalida la memòria i les torna a descarregar automàticament. Tot seguit verifica la signatura RS256 i comprova l'emissor, l'audiència i la caducitat del token. Aquesta validació es resol en una dependència reutilitzable que s'injecta a cada *endpoint* protegit, separant deliberadament la verificació del token de la cerca

de l'usuari a la base de dades: això permet que, en el primer accés, l'*endpoint* de perfil apliqui un patró *get-or-create* -crea l'usuari a partir dels *claims* del token si no existeix i, si ja existeix, en retorna el perfil-, mentre que la resta d'operacions exigeixen un usuari ja registrat. El token s'emmagatzema de manera segura al dispositiu, i el perfil és editable: el nom visible i l'idioma preferit es modifiquen des de l'aplicació.

Pel que fa a l'idioma, el sistema distingeix dos nivells que conviuen sense interferir. D'una banda, cada llar té un **idioma canònic** (*default_language*), que es fixa en el moment de crear-la -amb l'idioma preferit del creador com a valor per defecte- i que determina l'idioma en què s'emmagatzemen i s'indexen totes les dades de l'inventari; aquesta és la peça que garanteix un BM25 consistent dins la llar, ja que les entitats sempre s'hi normalitzen independentment de com s'hagin expressat. D'altra banda, cada usuari té un **idioma preferit** propi, que és el de la interfície i el de les respostes que rep, i que pot canviar des del perfil; aquest canvi sincronitza el *pipeline* NLP amb la interfície, de manera que la mateixa llar pot ser utilitzada per membres amb idiomes diferents -cadascú interactuant i rebent respostes en el seu- mentre les dades subjacents es mantenen en un únic idioma comú.

La tria de fixar aquest idioma canònic **per llar** -i no un idioma global únic per a tot el sistema- mereix una justificació, perquè totes dues opcions garantirien la consistència del BM25. La diferència no és de cost sinó de **fidelitat de les dades**. Com que el cas habitual és que tots els membres d'una mateixa llar parlin el mateix idioma, amb l'enfocament per llar la traducció esdevé l'**excepció** i no la norma: el camí comú és una simple extracció d'entitats sense pèrdua, i només cal traduir quan un membre s'expressa en un idioma diferent del de la llar. Convé remarcar que això **no estalvia crides** al model de llenguatge -la normalització es resol dins la mateixa crida d'interpretació que es faria igualment-; el guany és exclusivament de qualitat. Enfront d'emmagatzemar-ho tot en un idioma global mitjançant una traducció sistemàtica, l'opció per llar **manté les dades llegibles en la llengua pròpia de la casa, acota el risc de traducció incorrecta** al cas minoritari en lloc de fer-lo sistemàtic sobre l'emmagatzematge canònic, i **no cedeix qualitat de cerca**: les consultes sempre van filtrades per llar i mai no creuen els seus límits, i com que l'índex no fa servir un analitzador específic d'idioma sinó el per defecte -i els *embeddings* i el *reranker* d'Azure són multilingües-, unificar-ho tot en un sol idioma amb prou feines milloraria la recuperació i quedaria contrarestat pel risc de traducció.

5.6 Accessibilitat i experiència (clau AAL)

L'orientació a persones grans i a l'àmbit AAL es concreta en el **Mode Senior**, amb tipografia, botons i icones més grans, i una pantalla d'assistent a pantalla completa amb botons grans on el VAD cobreix el tancament automàtic de l'escolta. L'aplicació està localitzada en català, castellà i anglès, amb prop de 140 claus de traducció i l'idioma inicial derivat del dispositiu. Tot plegat se sosté sobre un sistema de disseny propi, *Warm Home*, amb un conjunt de *tokens* (color, ombra, radis, tipografia, moviment) i divuit *widgets* compartits.

A aquesta orientació s'hi sumen dos detalls d'experiència que faciliten l'ús quotidià. El primer són les **cerques re-cents**: la pantalla de cerca recorda les darreres cerques de l'usuari i les ofereix com a accés ràpid. L'historial es desa **localment** al dispositiu i està delimitat **per usuari i per llar**, de manera que cada combinació de compte i llar té la seva pròpia llista independent; així, en un mòbil compartit l'historial no es filtra entre comptes i, dins d'un mateix compte, només es mostren les cerques de la llar que s'està consultant -coherent amb el fet que les cerques s'executen contra l'inventari d'una llar concreta-. Una cerca només s'afegeix a l'historial quan l'usuari **obre un resultat**, no a cada pulsació ni en prémer cercar, de manera que la llista recull únicament les consultes realment útils; desduplica sense distingir majúscules, manté l'ordre de més recent a menys i es limita a vuit entrades, i es rehidrata automàticament quan l'usuari canvia de llar o de compte. El segon és la **selecció automàtica de la icona d'habitació**: quan l'usuari crea una habitació sense triar-ne icona, el sistema n'infereix una a partir del nom mitjançant coincidència de paraules clau multilingüe (català, castellà i anglès) -"cuina", "cocina" o "kitchen" reben la icona de coberts; "bany" o "bathroom", la de dutxa, etc.-. Aquesta icona inferida actua només com a valor per defecte: si l'usuari n'escull una d'explícita des del detall de l'habitació, aquesta preval. D'aquesta manera l'inventari es mostra amb icones coherents sense exigir cap configuració, alhora que respecta l'elecció manual quan n'hi ha.

5.7 Infraestructura cloud i operació

Tota la plataforma s'executa sobre Azure: Container Apps i Container Registry per al desplegament, PostgreSQL 16 Flexible Server, AI Search, Azure OpenAI, Speech, Key Vault amb *Managed Identity* i Application Insights. La seguretat dels secrets és estricta: tot resideix a Key Vault i es llegeix mitjançant identitat administrada, sense aparèixer mai en codi ni en *logs*. Les migracions de la base de dades es gestionen amb Alembic en mode asíncron, i s'han desenvolupat *scripts* de *smoke test* i de diagnòstic. Cal destacar que, gràcies a l'ús de *tiers* gratuïts i de consum, el cost mensual en producció es manté per sota dels deu euros. El procés de signatura digital de l'APK, la seva importància per a la distribució en dispositius reals i la configuració condicional es detallen a l'Apèndix 8.

5.8 Model de seguretat

Tot i que la seguretat travessa la resta d'apartats, val la pena recollir-la de manera unificada, perquè el sistema l'aborda en diverses capes que es complementen. La primera és l'**autenticació**, ja descrita a l'apartat 5.5: cada petició porta un JWT verificat criptogràficament (RS256) contra les claus públiques d'Entra. Sobre ella se situa l'**autorització**, que distingeix dos nivells dins de cada llar -membre i propietari-: les operacions de lectura exigeixen ser membre, mentre que les administratives o destructives (convidar, expulsar, transferir la propietat o esborrar la llar) queden restringides al propietari. La tercera capa és l'**aïllament multi-tenant**: totes les consultes filtren per *home_id*, també les que van a Azure AI Search, de manera que un usuari mai no pot arribar a dades d'una llar de la qual no

és membre. A aquestes s'hi afegeixen la **gestió de secrets** -centralitzada a Key Vault i llegida per *Managed Identity*, com s'ha vist a l'apartat 5.7-, la seguretat del **transport** -SSL obligatori cap a PostgreSQL i HTTPS en tots els serveis d'Azure- i, com a última línia de defensa, la **validació de dades**: Pydantic comprova tipus, formats i límits de longitud de totes les entrades a l'entrada de l'API, i la base de dades reforça la integritat amb *constraints* CHECK i UNIQUE. La combinació d'aquestes capes fa que cap petició no pugui ni saltar-se l'autenticació, ni accedir a dades alienes, ni introduir dades malformades sense ser rebutjada.

6 CONCLUSIÓ

Contrastant l'estat actual del desplegament amb els objectius fixats a l'informe inicial, es pot afirmar que el projecte ha assolit el seu objectiu general, ja que s'ha desenvolupat una aplicació mòbil d'assistència a la vida diària, funcional d'extrem a extrem, que permet gestionar l'inventari domèstic mitjançant llenguatge natural per veu i text. El sistema s'articula sobre una arquitectura *Retrieval-Augmented Generation* real -recuperació híbrida sobre Azure AI Search combinada amb un model generatiu (gpt-4.1)- i sobre un model multi-tenant amb llars compartides entre diversos usuaris. El nucli conceptual del TFG, centrat en l'ús del llenguatge natural per part de l'usuari, que s'expressa lliurement, està assolit: el sistema interpreta la seva intenció, actua sobre les seves dades i respon en llenguatge natural, i no només està codificat sinó verificat amb dades reals.

Pel que fa als objectius específics, es troben assolits i verificats. L'anàlisi i el model de dades jeràrquic (usuari → habitatge → habitació → contenidor → objecte) està plenament implementat a la base de dades amb les seves taules i el mecanisme de compartició via invitacions amb estats i caducitat. El disseny arquitectònic Frontend-Backend-LLM-Base de dades s'ha materialitzat amb Flutter, FastAPI, Azure OpenAI i PostgreSQL més AI Search, i el flux RAG funciona sobre una cerca híbrida BM25 + vector + *re-ranking* semàntic. El backend és complet i s'ha validat amb diferents tests, verificant tot el CRUD i l'autenticació JWT contra el tenant CIAM. La integració amb el LLM compleix els tres requisits plantejats -prompts robustos, sortides JSON estructurades i desambiguació semàntica mitjançant targetes de clarificació quan la confiança de la intenció és baixa-. El frontend mòbil incorpora les nou funcionalitats previstes, l'entrada per veu i text via Azure Speech, la gestió de comptes, llars i invitacions, i una vessant d'accessibilitat concreta a través del Mode Senior.

Per altra banda, la validació i avaluació s'ha cobert de forma completa: s'han fet tests funcionals i proves amb escenaris reals -login real des d'Android i pipeline NLP d'extrem a extrem-, i s'ha testejat el funcionament de l'aplicació en una llar amb múltiples usuaris reals durant un període d'ús quotidià.

Més enllà del simple compliment d'objectius, en aquest punt del projecte podem extreure algunes conclusions. La primera és que el diferencial respecte a l'estat de l'art s'ha materialitzat de manera efectiva: a diferència de propostes com *MemPal*, basada en una càmera portable, o *HomeBox*, orientada a la catalogació estàtica, el sistema funciona

sense sensors ni dispositius externs, únicament amb el tel·lèfon mòbil i el llenguatge natural, i amb una gestió dinàmica i compartida de la informació. La premissa de partida, doncs, s'ha demostrat viable. La segona és que les decisions arquitectòniques han resistit la implementació real i s'han mostrat sostenibles, amb un cost mensual en producció inferior als deu euros gràcies a l'ús de *tiers* gratuïts i de consum.

Com a línies de continuació, el treball futur passa de manera natural per afrontar objectius i idees que estaven en ment en una fase inicial del projecte: implementar el mòdul de recomanacions intel·ligents o la inserció de l'inventari de forma automàtica mitjançant visió per computador.

Una altra línia de continuïtat és la normalització singular-plural en l'agrupació d'objectes: actualment dues files només es fusionen si el nom coincideix de manera exacta, de manera que un “boli” no s'agrupa amb uns “bolis” existents. No és trivial perquè el sistema és multilingüe (cada idioma té plurals irregulars) i el risc real és fusionar de més amb paraules invariables o intrínsecament plurals (parai-gües, estisores). L'opció preferida -per coherència amb l'arquitectura- és canonitzar el nom a singular en la interpretació del llenguatge natural; mentrestant, la recomanació d'ús és crear sempre els objectes en singular.

ÚS DE LA IA GENERATIVA

La intel·ligència artificial generativa ha tingut un doble paper en aquest treball, i convé descriure'l de manera transparent. D'una banda, com a **component del sistema**: el producte mateix depèn d'un model de llenguatge (gpt-4.1) per interpretar la intenció de l'usuari, extreure'n les entitats i generar les respostes en llenguatge natural, i d'un model d'*embeddings* (text-embedding-3-small) per a la cerca semàntica. En aquest cas, l'ús de la IA és el nucli funcional del projecte i s'ha tractat com a tal. Això s'ha materialitzat, en primer lloc, en un **disseny iteratiu dels prompts** (punt 5.1): no com a redacció puntual, sinó com un contracte d'instruccions afinat provant el comportament real del model, on diverses regles són la resposta a errors concrets observats i on l'elecció de la formulació i de la temperatura condiciona directament la fiabilitat de la sortida. Sobre aquesta base s'hi afegeixen les sortides estructurades i verificables mitjançant *Structured Outputs*, els mecanismes de desambiguació davant la baixa confiança i la calibració empírica dels llindars de decisió.

D'altra banda, la IA generativa s'ha utilitzat com a **eina de suport durant el desenvolupament**: assistència en la programació, en la documentació tècnica i en la redacció i estructuració d'aquest mateix informe. En tots aquests casos, l'ús ha estat supervisat: el codi generat o suggerit s'ha revisat, provat i integrat manualment, i els continguts redactats amb assistència s'han verificat i adaptat perquè reflecteixin l'estat real del projecte i la veu de l'autor. Cap dada, mètrica ni resultat presentat en aquest document prové de la IA sense una verificació humana prèvia contra les fonts reals del projecte.

AGRAÏMENTS

Vull agrair al meu tutor el seguiment i les orientacions al llarg del projecte, així com a la família i a les persones properes que han posat a prova l'aplicació.

BIBLIOGRAFIA

- [1] HomeBox. (2024). *Home inventory management application*. Disponible a: <https://homebox.software/en/>
- [2] MIT Media Lab. (2025). *MemPal: Wearable Memory Assistant for Aging Population*. Disponible a: <https://www.media.mit.edu/projects/mempal/overview/>
- [3] Augusto, J. C., et al. (Eds.) (2012). *Handbook of Ambient Assisted Living*. IOS Press.
- [4] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., et al. (2020). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS).
- [5] Brown, T., Mann, B., Ryder, N., et al. (2020). *Language Models are Few-Shot Learners*. Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS).
- [6] Schwaber, K., and Sutherland, J. (2020). *The Scrum Guide: The Definitive Guide to Scrum*. Scrum.org.
- [7] Ramírez, S. (2024). *FastAPI Documentation*. Disponible a: <https://fastapi.tiangolo.com>
- [8] SQLAlchemy. (2024). *SQLAlchemy 2.0 Documentation - Asyncio Support*. Disponible a: <https://docs.sqlalchemy.org/en/20/orm/extensions/asyncio.html>
- [9] Pydantic. (2024). *Pydantic Documentation*. Disponible a: <https://docs.pydantic.dev>
- [10] Microsoft. (2024). *Hybrid search and Reciprocal Rank Fusion (RRF) in Azure AI Search*. Microsoft Learn. Disponible a: <https://learn.microsoft.com/azure/search/hybrid-search-ranking>
- [11] Microsoft. (2024). *Semantic ranking in Azure AI Search*. Microsoft Learn. Disponible a: <https://learn.microsoft.com/azure/search/semantic-search-overview>
- [12] Microsoft. (2024). *What is Microsoft Entra External ID*. Microsoft Learn. Disponible a: <https://learn.microsoft.com/entra/external-id/>
- [13] OpenAI. (2024). *Structured Outputs*. OpenAI Platform Documentation. Disponible a: <https://platform.openai.com/docs/guides/structured-outputs>
- [14] Google. (2024). *Flutter Documentation*. Disponible a: <https://docs.flutter.dev>
- [15] Robertson, S., and Zaragoza, H. (2009). *The Probabilistic Relevance Framework: BM25 and Beyond*. Foundations and Trends in Information Retrieval, 3(4), 333–389.
- [16] Cormack, G. V., Clarke, C. L. A., and Büttcher, S. (2009). *Reciprocal Rank Fusion Outperforms Condorcet and Individual Rank Learning Methods*. Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval.
- [17] Malkov, Yu. A., and Yashunin, D. A. (2020). *Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 42(4), 824–836.
- [18] Nogueira, R., and Cho, K. (2019). *Passage Re-ranking with BERT*. arXiv preprint arXiv:1901.04085.
- [19] Sakimura, N., Bradley, J., and Agarwal, N. (2015). RFC 7636: *Proof Key for Code Exchange by OAuth Public Clients (PKCE)*. Internet Engineering Task Force (IETF).

APÈNDIX

A1. Justificació de l'stack tecnològic

L'elecció de l'*stack* es justifica per la naturalesa fortament orientada a entrada/sortida del Backend. S'ha triat **FastAPI** [7] sobre Python perquè el seu model asíncron natiu permet encadenar de manera concurrent les múltiples crides d'E/S que requereix cada petició (base de dades, Azure OpenAI i AI Search) sense bloquejar el servidor, i perquè la seva integració amb **Pydantic** [9] aporta validació estricta de tipus i límits a totes les entrades i sortides amb molt poc codi. La persistència s'ha construït amb **PostgreSQL** i **SQLAlchemy 2.0** [8] en mode asíncron sobre el *driver* `asyncpg`, mantenint la coherència asíncrona de tota la pila. El desplegament es fa amb contenidors Docker sobre **Azure Container Apps**, un servei *serverless* que escala segons la demanda i s'autentica contra Key Vault i AI Search mitjançant *Managed Identity*, sense haver de gestionar claus addicionals. Aquestes eleccions, preses a l'inici, han resistit tot el desenvolupament sense necessitat de revertir-les.

A2. Camps de l'esquema JSON de la interpretació

El model retorna en cada interpretació una estructura JSON amb un contracte de camps fix. Més enllà de la intenció (*intent*) i de les entitats principals -l'objecte (*object*) i la ubicació de destí (*location*)-, l'esquema incorpora els camps que doten de robustesa el *pipeline*. El camp `from_location` recull la ubicació d'origen i només s'emplega en els moviments, de manera que el sistema pot distingir d'on surt un objecte i cap a on va. El camp `quantity` captura la quantitat únicament quan l'usuari l'esmenta de manera explícita -en cas contrari queda a nul- i és el que alimenta la lògica de moure o eliminar per unitats (apartat 5.2). El camp `confidence`, un valor entre 0 i 1, expressa el grau de certesa del model sobre la seva pròpia interpretació. Els camps `needs_clarification` (booleà) i `clarification_options` (llista d'opcions) governen la desambiguació a nivell de llenguatge: el model els activa quan la instrucció és intrínsecament ambigua o incompleta -per exemple, "*mou això*"- per demanar més informació sense executar cap acció. Finalment, el camp `message` ja conté una resposta en llenguatge natural en l'idioma de l'usuari, que actua com a xarxa de seguretat del sistema si la generació de resposta falla.

A3. Detall tècnic del cicle de captura de veu

El cicle de captura de veu es governa amb diversos mecanismes de tancament complementaris. El reconeixement treballa sobre un flux d'àudio PCM de 16 bits, 16 kHz i mono que s'envia per blocs al servei mentre l'usuari parla, i la transcripció parcial es mostra en temps real. El tancament del micròfon es decideix, en primer lloc, pel VAD del propi servei, que detecta el final de la parla quan es fa el silenci; en segon lloc, per la parada manual de l'usuari, que pot tocar el micròfon per finalitzar; i, com a xarxa de seguretat, per un *fallback* temporal de **15 segons** que tanca la captura si cap dels altres dos no s'ha disparat -per exemple, si l'usuari no arriba a parlar-. Un cop tancat el micròfon, el sistema concedeix un marge addicional de **5 segons** perquè

el servei retorni la transcripció final; si en aquest marge no arriba res, la captura s'avorta de manera neta sense bloquejar la interfície.

A4. Continuïtat del diàleg i coherència de la interfície

Tot i que la conversa amb el Backend és *stateless* -cada petició és independent-, el client manté la **continuïtat del diàleg** reenviant a cada crida una finestra dels darrers torns: concretament, els **últims cinc intercanvis (deu missatges, alternant usuari i assistent)**, que es purguen per la part més antiga a mesura que la conversa creix. Aquesta finestra permet resoldre referències que depenen del context immediat -"*i les ulleres?*" després d'una consulta prèvia-sense que el Backend hagi de persistir cap sessió, i es buida en tancar la conversa (per cancel·lació, descartament o esgotament del temps sense cap espera activa), de manera que cada nou diàleg parteix net.

Un darrer detall dona solidesa a la interacció: la **coherència immediata de la interfície**. Com que l'assistent muta les dades directament al Backend -i no per la mateixa via que fan servir les pantalles convencionals-, les vistes podrien quedar desactualitzades fins que l'usuari canviés de pantalla. Per evitar-ho, després de cada resposta que ha modificat dades el client invalida selectivament la memòria cau de les pantalles afectades (objectes, habitacions i contenidors) perquè es tornin a demanar. La condició és precisa: només es refresca quan la resposta inclou una acció que realment muta -`insert`, `move`, `remove`, `create_room` o `create_container`-, mentre que les consultes (`query`, `history`) i les aclaracions no disparen cap recàrrega innecessària. Així, dir "*mou les claus al calaix*" per veu actualitza també, a l'instant, la pantalla que hi ha al darrere.

A5. Model de dades relacional

Aquesta jerarquia es materialitza en un model de dades relacional de deu taules, quatre tipus enumerats, set *triggers* i dues vistes, que recull tant l'inventari com la capa multi-tenant. Al voltant de l'usuari i la llar s'articulen les habitacions, els contenidors -amb una referència a si mateixos (`parent_container_id`) que permet l'arbre recursiu- i els objectes; la compartició s'expressa amb `home_members` (rol de cada usuari dins la llar) i les invitacions, i el comportament dinàmic del sistema queda registrat a `object_aliases` (noms alternatius d'un objecte), `object_movements` (historial d'insercions, moviments i eliminacions) i `search_index_metadata`. Aquesta darrera taula mereix una menció especial perquè és la que sosté la coherència entre les dues fonts de dades: hi desa l'estat de sincronització de cada objecte cap a Azure AI Search (`synced`, `pending` o `error`) juntament amb el darrer error, de manera que un document que ha fallat en indexar-se es pot reintentar i mai no queda silenciosament desincronitzat respecte a la base de dades.

Sobre els objectes pendents d'assignar opera una asimetria deliberada: el cercador manual els **oculta** -ja tenen la seva secció dedicada a la pantalla d'inici-, però l'assistent de veu **sí que els localitza**, de manera que l'usuari pot reassignar-los, consultar-los o eliminar-los simplement parlant -"*posa*

el martell al traster" - sense haver d'entrar a la vista de pendents. Tots dos camins comparteixen el mateix motor de cerca; l'única diferència és un indicador que el cercador activa per excloure els objectes sense ubicació i que l'assistent desactiva precisament per poder gestionar-los.

A6. Calibració i banc d'avaluació del reranker

Val la pena fer una observació sobre el sostre d'aquesta escala, perquè condiciona la manera de fixar els llindars. Tot i que el reranker puntua de 0 a 4, a la pràctica les cerques del sistema no s'acosten mai al valor màxim: fins i tot els encerts més clars es queden en una franja estreta, força per sota del 4 -els senyals legítims observats al banc van d'1,8 fins a 3,1 (el màxim observat), mentre que el gruix dels objectes inexistents no supera 1,73-. El motiu és que el 4 representa, per al model, una coincidència semàntica completa i rica entre una consulta i un passatge de text extens i informatiu, que és el tipus de dada amb què aquests rerankers van ser entrenats. En el nostre cas, en canvi, tant la consulta com el document indexat són molt breus: la consulta sol ser el nom d'un objecte o una frase curta, i el document és un registre denormalitzat i concís (nom, descripció i ubicació), sense la densitat textual que permetria saturar l'escala. Per tant, el sostre efectiu del nostre domini no és el 4 teòric sinó un valor sensiblement més baix, i això explica per què la calibració no busca la proximitat absoluta al màxim sinó la **separació relativa dins d'aquesta banda comprimida**: el que distingeix un encert d'un fals positiu no és com de prop està del 4, sinó si supera el mínim de rellevància i si se separa prou del segon candidat. Reconèixer aquesta limitació de l'escala, lluny de ser un inconvenient, és el que ha permès afinar els llindars sobre el rang real d'operació i no sobre un màxim que el sistema mai assolirà.

La Fig. 5 mostra la distribució empírica de les puntuacions del reranker semàntic sobre el banc d'avaluació real, i permet visualitzar la justificació del llindar mínim escollit de 1,8.

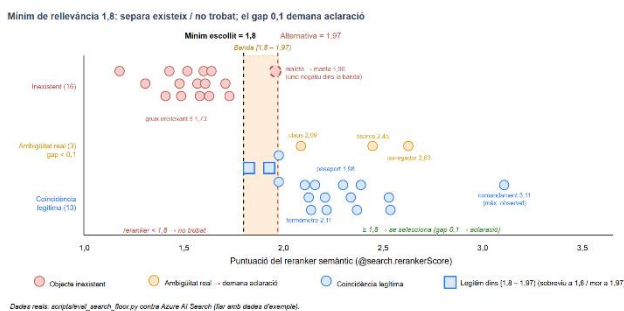


Fig. 5. Distribució empírica de les puntuacions del reranker semàntic i justificació del llindar mínim de 1,8. El gràfic representa les puntuacions del reranker semàntic (@search.rerankerScore) per a tres categories de cerca: objectes inexistents (cercles rosats, 16 casos), ambigüitats reals en què el gap entre els dos primers candidats és inferior a 0,1 i el sistema ha de demanar aclaració (cercles grocs, 3 casos), i coincidències legítimes en què el sistema identifica l'objecte de manera directa (cercles blaus, 13 casos). El gruix dels objectes inexistents es concentra per sota

de 1,73, mentre que les coincidències legítimes se situen per sobre de 1,8. El llindar mínim de 1,8 (línia discontinua negra) separa nítidament els dos grups, amb l'única excepció del cas "maleta" → "manta" (1,96), l'únic fals positiu dins la banda 1,8–1,97, representat per un quadrat blau. Augmentar el llindar fins a 1,97 (línia discontinua vermella) eliminaria les coincidències legítimes situades en aquesta banda, provocant falsos negatius en casos vàlids a canvi de resoldre únicament aquest cas atípic. L'anàlisi vàlida que el llindar de 1,8 és l'opció òptima: manté totes les coincidències reals i accepta com a única excepció un fals positiu excepcional, mentre el gap de 0,1 s'ocupa de distingir les ambigüitats genuïnes de les coincidències unívokes.

A7. Indicador de confiança

Aquesta mateixa puntuació del reranker alimenta, a la interfície de cerca manual, un **indicador de confiança** que tradueix un valor tècnic poc intuïtiu en un percentatge comprensible. El càlcul es desacobla deliberadament del valor cru: combina el **nivell absolut** del millor resultat -mitjançant una finestra *smoothstep* entre 1,6 i 2,3 que actua de sostre compartit- amb la **proximitat** de cada resultat a aquest millor candidat, modelada amb un decaïment exponencial $\exp((x - \text{top}) / 0,35)$; el percentatge final es limita a la franja d'1 a 99 %. El detall rellevant és que la normalització es fa **respecte al màxim i no respecte a la suma**: una normalització de suma (tipus *softmax*) repartiria la massa entre els resultats, de manera que dues coincidències igual de bones -cercar "claus" en una llar amb dues entrades reals- es quedarien cadascuna en un percentatge baix i enganyós; ancorant en el millor resultat, en canvi, un empat conserva gairebé tota la confiança i tots dos resultats es lleixen alts pel seu propi mèrit. El sòl d'aquesta finestra (1,6) se situa a propòsit per sota del mínim de rellevància del Backend (1,8), perquè tots dos viuen en *pipelines* diferents sobre distribucions diferents: el cercador rep paraules soltes -amb un reranker més comprimit-, mentre que l'1,8 es va calibrar sobre les frases normalitzades de l'assistent. Quan no hi ha reranker (cerca no rankejada semàntica), no es mostra cap percentatge, evitant així una confiança falsa.

A8. Signatura digital i distribució de l'APK

Perquè l'aplicació es pogués repartir i provar en dispositius reals -i no només a l'emulador de desenvolupament-, l'APK de *release* es **signa amb un magatzem de claus (*keystore*)** propi generat per al projecte, en lloc de la clau de depuració per defecte de Flutter. Android exigeix que tot APK estigui signat digitalment i, sobretot, que **cada actualització es signi amb la mateixa clau**: el sistema operatiu rebutja instal·lar una versió nova sobre una d'existent si les signatures no coincideixen. Disposar d'un *keystore* estable i custodiat és, doncs, el que permet **lliurar versions successives** de l'aplicació al llarg del temps sense desinstal·lar-la ni perdre'n les dades. La configuració de compilació carrega les credencials del *keystore* de manera condicional: si hi són, signa la *release* amb la clau pròpia; si no -per exemple, en un clon del repositori sense la clau-, recau de manera segura en la signatura de depuració. Com que aquest APK ja apunta al Backend de producció a Azure,

funciona de manera autònoma en qualsevol mòbil per *sideload*, sense dependre de l'entorn de desenvolupament. Aquesta és, precisament, la peça que ha fet possible la validació en una llar amb diversos usuaris reals durant un període d'ús quotidià que es recull a les conclusions (apartat 6).