



**Universitat**  
de les Illes Balears

## **TREBALL DE FI DE GRAU**

# **DISSENY D'UN ASSISTENT BASAT EN IA INTEGRAT EN L'ENTORN EMPRESARIAL DE VISMEL**

**Marc Llobera Villalonga**

**Enginyeria Informàtica**

**Escola Politècnica Superior**

**Any acadèmic 2025-26**



# **DISSENY D'UN ASSISTENT BASAT EN IA INTEGRAT EN L'ENTORN EMPRESARIAL DE VISMEL**

**Marc Llobera Villalonga**

**Treball de Fi de Grau**

**Escola Politècnica Superior**

**Universitat de les Illes Balears**

**Any acadèmic 2025-26**

Paraules clau del treball: Assistent Conversacional, IA, OpenAI API, Enginyeria de Prompt

*Tutor: Antonio Bibiloni Coll*



Agraeixo profundament a la meva família, especialment als meus pares, pel seu suport constant desinteressat. També a la meva parella, per la seva paciència i companyia durant les llargues hores dedicades a aquest treball. Finalment, vull agrair als professors que han convertit el meu futur com a informàtic en una veritable vocació.



# SUMARI

<b>Sumari</b>	<b>iii</b>
<b>Índex de figures</b>	<b>v</b>
<b>Índex de taules</b>	<b>vii</b>
<b>Acrònims</b>	<b>ix</b>
<b>Resum</b>	<b>xi</b>
<b>1 Introducció</b>	<b>1</b>
1.1 Context i Motivació . . . . .	1
1.2 Objectius . . . . .	2
1.3 Abast i Limitacions . . . . .	2
1.4 Metodologia . . . . .	3
1.5 Estructura del Document . . . . .	4
<b>2 Marc Teòric</b>	<b>5</b>
2.1 Introducció al Marc Teòric . . . . .	5
2.2 Antecedents i Estat de l'Art . . . . .	5
2.3 Conceptes Clau . . . . .	6
2.3.1 <i>Large Language Model</i> (Model de Llenguatge Gran) (LLM) . . .	6
2.3.2 Intel·ligència artificial, Aprenentatge Automàtic i Processament de Llenguatge Natural . . . . .	7
2.4 Plantejament del Problema . . . . .	7
2.5 Tecnologies i Eines . . . . .	8
2.5.1 Entorn de Desenvolupament . . . . .	9
2.5.2 Tecnologies i Llenguatges . . . . .	9
2.5.3 Models de Llenguatge i Selecció de l'Assistent . . . . .	10
2.6 Arquitectura del Sistema . . . . .	14
2.7 Seguretat i Privacitat . . . . .	15
<b>3 Disseny i Requisits</b>	<b>17</b>
3.1 Introducció al Disseny i Requisits del Projecte . . . . .	17
3.2 Requisits del Sistema . . . . .	17
3.2.1 Requisits Funcionals . . . . .	17
3.2.2 Requisits No Funcionals . . . . .	18
3.3 Disseny de la Base de Dades . . . . .	19

3.3.1	Base de Dades PostgreSQL . . . . .	19
3.3.2	Base de Dades Redis . . . . .	25
3.4	Interfície Gràfica . . . . .	26
3.5	Comunicació entre Mòduls . . . . .	29
3.5.1	Frontend -> Backend Node.js . . . . .	29
3.5.2	Backend Node.js -> Backend Python . . . . .	30
3.5.3	Flux de Comunicació amb l'LLM d'OpenAI . . . . .	31
3.6	Estructura de Carpetes del Projecte . . . . .	40
3.6.1	Carpeta Frontend . . . . .	41
3.6.2	Carpeta Backend Node . . . . .	42
3.6.3	Carpeta Backend Python . . . . .	43
3.6.4	Carpeta de Bases de Dades . . . . .	44
<b>4</b>	<b>Implementació</b>	<b>47</b>
4.1	Introducció a la Implementació . . . . .	47
4.2	Entorn de Desenvolupament i Producció . . . . .	47
4.3	Implementació dels Mòduls . . . . .	52
4.3.1	Frontend (Vite + React) . . . . .	52
4.3.2	Backend Node.js . . . . .	57
4.3.3	Backend Python . . . . .	63
4.4	Seguretat en la Implementació . . . . .	66
4.5	Dificultats i Solucions Adoptades . . . . .	68
<b>5</b>	<b>Proves i Validació</b>	<b>75</b>
5.1	Introducció a les Proves . . . . .	75
5.2	Estratègia de les Proves . . . . .	75
5.3	Resultats de les Proves Realitzades . . . . .	76
5.4	Conclusions i Limitacions . . . . .	79
<b>6</b>	<b>Conclusions i Treball Futur</b>	<b>81</b>
<b>A</b>	<b>Annexos</b>	<b>83</b>
A.1	Codi Frontend . . . . .	83
A.2	Codi Backend Node . . . . .	85
A.3	Codi Backend Python . . . . .	86
A.4	Llistat de Preguntes i Respostes del Xatbot . . . . .	92
A.4.1	Preguntes Generades per l'Autor . . . . .	92
A.4.2	Preguntes Formulades per un Usuari de l'Empresa . . . . .	103
	<b>Glossari</b>	<b>107</b>
	<b>Bibliografia</b>	<b>109</b>



## ÍNDIX DE FIGURES

2.1	Diagrama de fluxe dels mòduls del projecte . . . . .	14
3.1	Exemple taules de la Base de Dades que usava Vismel . . . . .	20
3.2	Diagrama de la Base de Dades . . . . .	21
3.3	Pàgina d'inici de sessió . . . . .	26
3.4	Pàgina Home . . . . .	27
3.5	Pàgina Dashboard: Elements 1 i 2 . . . . .	27
3.6	Pàgina Dashboard: Elements 3 i 4 . . . . .	28
3.7	Pàgina Chatbot . . . . .	28
3.8	Desplegable amb Funcions de l'Usuari . . . . .	29
3.9	Pàgina Users . . . . .	29
3.10	Diagrama de Flux del Backend de Python . . . . .	32
3.11	Estructura d'Arxius: Arrel . . . . .	41
3.12	Estructura d'Arxius: Frontend . . . . .	42
3.13	Estructura d'Arxius: Backend Node . . . . .	43
3.14	Estructura d'Arxius: Backend Python . . . . .	44
3.15	Estructura d'Arxius: Bases de Dades . . . . .	45



## ÍNDIX DE TAULES

2.1	Comparació Models OpenAI . . . . .	12
3.1	Columnes de la taula usuario . . . . .	21
3.2	Columnes de la taula rol . . . . .	22
3.3	Columnes de la taula maquina . . . . .	22
3.4	Columnes de la taula tipo_maquina . . . . .	22
3.5	Columnes de la taula modelo_maquina . . . . .	22
3.6	Columnes de la taula frecuencia_revision . . . . .	23
3.7	Columnes de la taula revision_maquina . . . . .	23
3.8	Columnes de la taula tipo_revision . . . . .	23
3.9	Columnes de la taula recaudacion_semanal . . . . .	24
3.10	Columnes de la taula averia . . . . .	24
3.11	Columnes de la taula cliente . . . . .	24
3.12	Columnes de la taula poblacion . . . . .	25
3.13	Columnes de la taula articulo . . . . .	25
3.14	Columnes de la taula familia_articulo . . . . .	25
3.15	Columnes de la taula entradas_salidas . . . . .	25



## ACRÒNIMS

**TFG** Treball Final de Grau

**IA** Intel·ligència Artificial

**BD** Base de Dades

**LLM** *Large Language Model* (Model de Llenguatge Gran)

**JS** JavaScript

**API** *Application Programming Interface* (Interfície de Programació d'Aplicacions)

**SQL** *Structured Query Language* (Llenguatge de Consulta Estructurat)

**UI** *User Interface* (Interfície d'Usuari)

**UX** *User Experience* (Experiència d'Usuari)

**HTTP** *Hypertext Transfer Protocol* (Protocol de Transferència d'Hipertext)

**HTTPS** *Hypertext Transfer Protocol Secure* (Protocol de Transferència d'Hipertext Segur)

**SGBD** Sistema Gestor de Bases de Dades

**CSS** *Cascading Style Sheets*

**HTML** *Hypertext Markup Language* (Llenguatge de Marcatge d'Hipertext)

**CPU** *Central Processing Unit* (Unitat Central de Processament)

**JWT** *JSON Web Token*

**TPM** Tokens Per Minut

**ML** *Machine Learning* (Aprenentatge Automàtic)

**NLP** *Natural Language Processing* (Processament de Llenguatge Natural)

**POO** Programació Orientada a Objectes

**IG** Interfície Gràfica

**SO** Sistema Operatiu

**RAM** *Random Access Memory* (Memòria d'Accés Aleatori)

**REST** *Representational State Transfer* (Transferència d'Estat Representacional)

**IP** *Internet Protocol* (Protocol d'Internet)

**RAG** *Retrieval-Augmented Generation* (Generació Augmentada de Recuperació)

## RESUM

Vismel és una petita empresa de maquinària i productes de *vending* ubicada en el centre de Mallorca. Té clients per tota l'illa i de totes les índoles com ara hotels, residències d'ancians, edificis públics o col·legis, etc. Fundada fa molts d'anys, sempre ha emprat un *software* de gestió antic, amb poc manteniment i cap actualització significativa, tot i que encara funciona correctament.

Aquest Treball Final de Grau (TFG) consisteix en el desenvolupament d'un bot conversacional intel·ligent, que sorgeix com a part d'un projecte personal més ampli amb la creació d'un nou *software* enfocat a digitalitzar i automatitzar la majoria dels processos que es duen a terme a l'oficina de l'empresa.

El bot empra Intel·ligència Artificial (IA) mitjançant la OpenAI *Application Programming Interface* (Interfície de Programació d'Aplicacions) (API) i és capaç de respondre tota mena de consultes de manera similar als LLMs actuals, amb l'avantatge afegit d'accedir i utilitzar informació interna de l'empresa gràcies al context proporcionat per la Base de Dades (BD). Això es realitza mitjançant la generació automàtica de consultes sobre la base de dades, l'execució d'aquestes i la integració dels resultats dins la conversa, la qual cosa permet oferir respostes contextualitzades i específiques.

El resultat és una Web App desenvolupada principalment amb Python i JavaScript (JS), amb arquitectura separada de *backend* i *frontend*<sup>1</sup> i suport d'una base de dades *Structured Query Language* (Llenguatge de Consulta Estructurat) (SQL) i Redis. La pàgina inicial és un xat amb l'assistent, que simula serveis com ChatGPT, Gemini o Claude. L'usuari pot interactuar-hi fàcilment per obtenir informació rellevant com la recaptació de la setmana, les màquines que necessiten revisions, els articles a reposar, estimacions de venda, etc. Tot això està orientat a reduir la càrrega de treball, facilitar les tasques d'oficina i agilitzar l'accés a dades, especialment per a usuaris amb poca experiència tecnològica.

Durant el desenvolupament, ha estat clau l'aprenentatge i adaptació a noves tecnologies, especialment en àmbits com la creació d'APIs amb Node.js i la integració amb serveis d'IA a través de la OpenAI API.

---

<sup>1</sup> Els termes *frontend* i *backend* es refereixen a la separació funcional entre la capa de presentació (*frontend*) i la capa d'accés a les dades o interfície de gestió (*backend*).





## INTRODUCCIÓ

### 1.1 Context i Motivació

*Aquest projecte neix d'un vincle personal amb l'empresa Vismel, fundada fa dècades pel meu avi i gestionada actualment pels seus fills, entre ells la meua mare. Aquesta proximitat m'ha permès conèixer de primera mà les necessitats reals de l'empresa, especialment pel que fa als processos interns i la manca d'eines digitals modernes. En aquest context, el desenvolupament d'una eina per a l'optimització i digitalització de processos no només respon a un interès acadèmic, sinó també a una voluntat de contribuir a la modernització d'un entorn proper.*

Actualment, gran part de la seva operativa interna continua depenent d'un programari de gestió antic, desenvolupat a mida fa molts anys, que si bé compleix amb les funcionalitats bàsiques, presenta importants limitacions en termes de manteniment, escalabilitat, i actualització tecnològica. El sistema ha estat durant molt de temps l'eix central de les tasques administratives de l'empresa, entre les quals cal destacar la gestió d'inventari, registre de facturació i seguiment de màquines. Així i tot, no ofereix cap mena d'automatització avançada ni integració amb tecnologies modernes, i la seva interfície resulta antiquada. Emergeix així la necessitat d'una transformació digital que permeti modernitzar els processos interns i millorar l'eficiència general del negoci.

Aquesta eina consistiria en una Web App interna de l'empresa, accessible des de qualsevol ordinador de la nau. Mitjançant un sistema d'autenticació amb credencials, cada usuari podria accedir a un conjunt de funcionalitats específiques segons el seu rol assignat.

Aquestes funcionalitats no només millorarien les ja existents, sinó que també en permetrien d'altres de noves com ara l'organització de viatges de reposició segons els recursos disponibles, la gestió d'avaries, la visualització i anàlisi de dades, o la predicció de necessitats futures.

En conjunt, es tracta d'un projecte amb una visió a llarg termini, que requerirà manteniment i millora contínua. Per aquesta raó, aquest TFG se centra exclusivament en la implementació de la funcionalitat principal i més innovadora: una IA que actua com a xatbot empresarial, capaç d'accedir al context complet de les dades internes de l'empresa, amb l'objectiu d'assistir, principalment, als usuaris amb privilegis elevats en consultes operatives, agilitzar processos i millorar la presa de decisions.

### 1.2 Objectius

L'objectiu principal del TFG, amb el desenvolupament del bot, és proporcionar a l'empresa, i més concretament a la meua mare, una eina molt fàcil d'utilitzar per a qualsevol classe d'usuari. Aquesta eina ha de permetre consultar informació sobre les seves dades de manera senzilla i ràpida, sense necessitat de tenir coneixements especialitzats en informàtica ni entendre a fons un *software* per poder-ne fer un bon ús. A continuació, es detallen els objectius específics que es volen assolir amb aquest projecte:

- Permetre la realització de consultes complexes de manera senzilla, oferint respostes precises i entenedores dins dels seus límits. Això ha de facilitar la presa de decisions basada en dades i en anàlisis que, en altres circumstàncies, no es tindrien en compte.
- Millorar l'accessibilitat a les dades empresarials, proporcionant una *User Interface* (Interfície d'Usuari) (UI) moderna, de la mà amb el disseny actual de Vismel i que actuï com a pont entre l'usuari i la informació, evitant la necessitat de coneixements tècnics per accedir a dades rellevants.
- Garantir la privacitat i la seguretat de les dades, assegurant-se que el sistema implementi mesures adequades de protecció de la informació, d'acord amb les bones pràctiques actuals.
- Oferir una *User Experience* (Experiència d'Usuari) (UX) intuïtiva i eficient, prioritzant la simplicitat, la claredat en les respostes i la facilitat d'interacció amb el sistema per part de qualsevol classe d'usuari.

### 1.3 Abast i Limitacions

El projecte consta del disseny i desenvolupament de la funcionalitat principal del xatbot: integrat amb la IA de OpenAI, amb context personalitzat per l'entorn i els diferents tipus de decisions que pugui prendre i la capacitat de connexió directa amb la BD per executar consultes.

S'ha desenvolupat un sistema *backend* amb diferents APIs per comunicar-se amb el bot juntament amb un sistema d'autenticació mitjançant usuaris registrats i un sistema d'autorització per rols. Tot envoltat per una interfície web que comprèn totes les funcions.

L'aplicació s'ha dividit en diferents mòduls que poden córrer independentment l'un de l'altre gràcies a contenidors de Docker. Així podem trobar-hi:

- **BD PostgreSQL:** Correspon a la base de dades principal on es guarden tota la informació de l'empresa.
- **BD Redis:** S'utilitza com a memòria de *cache* compartida per guardar informació rellevant que no és necessari assegurar la seva persistència.
- **Backend amb Python:** Part encarregada de la connexió i conversació amb l'assistent usant la llibreria de OpenAI per la comunicació amb la seva API.
- **Backend amb Node.js:** Unifica totes les funcionalitats que es desenvoluparan en un futur. En aquest projecte principalment s'encarrega de ser el pont entre el *frontend* i el *backend* en Python, i de l'autenticació i autorització de l'aplicació.
- **Frontend amb Vite+React:** Dona interactivitat gràfica per les funcionalitats ofertes pels *backends*.

### 1.4 Metodologia

Es va començar per la definició clara de les tecnologies i les eines a utilitzar en cadascun dels mòduls del sistema. Per això, es va realitzar una recerca de les solucions més usades en el sector, prioritzant aquelles de caràcter Open Source (Codi Obert), amb facilitat d'ús, i escalabilitat a mitjà i llarg termini.

Un cop definides les tecnologies, es va procedir al disseny de la nova BD principal. Aquest procés va requerir no només crear una estructura optimitzada per a les necessitats actuals de l'aplicació, sinó també la migració de les dades emmagatzemades d'una BD antiga cap a la nova arquitectura. Aquesta transformació va implicar fer modificacions en els esquemes i netejar un volum considerable de dades, la qual cosa va ser una de les fases més llargues del projecte a causa de l'extensió de la informació tractada.

Amb la base de dades ja consolidada, el següent pas va consistir en el desenvolupament del mòdul central del sistema, el *backend* de Node.js. En aquesta etapa es va crear el servidor, es van configurar les variables d'entorn i es van establir les primeres APIs bàsiques juntament amb l'estructura general que posteriorment donaria suport a la resta de funcionalitats. També, es van desenvolupar els fitxers de connexió amb la BD i la lògica associada a aquestes APIs.

Posteriorment, es va crear el mòdul de Python, on es va implementar l'única ruta necessària per gestionar la interacció amb l'API d'OpenAI. Va incloure la creació de la connexió a la base de dades, la comunicació amb l'API externa i la definició del flux per generar respostes coherents i correctes a la consulta de l'usuari.

El desenvolupament va continuar amb la implementació del *frontend* orientat a garantir una UX fluida. Per fer-ho, es va seguir una lògica de navegació similar a la d'un usuari, començant amb el sistema d'inici de sessió, seguit per la barra de navegació, els components principals i finalment el xat del bot. Durant aquesta fase, les proves de les distintes funcionalitats varen donar noves necessitats a implementar com millores en el flux de gestió de la resposta del bot o la implantació de Redis per a la *cache*.

Finalment, es van ampliar les funcionalitats del *frontend* amb la creació de pàgines addicionals, com el dashboard i el mòdul de gestió d'usuaris, cosa que va requerir també la implementació de noves rutes al *backend*. Un cop l'aplicació va assolir un estat estable i complet, es va realitzar una fase de proves, en la qual es van identificar i corregir errors, a més d'incorporar millores per a la robustesa i usabilitat del sistema.

### 1.5 Estructura del Document

El present TFG s'ha estructurat en diversos capítols que responen a una progressió lògica, des de la motivació inicial fins a les conclusions i línies de futur.

El **Marc Teòric** (2) presenta els fonaments teòrics necessaris per entendre el projecte. Inclou els antecedents i l'estat de l'art, els conceptes clau relacionats amb els models de llenguatge i les tecnologies implicades, així com una descripció de les eines utilitzades, l'arquitectura proposada i les consideracions de seguretat i privacitat.

En **Disseny i Requisits** (3) es recull els requisits funcionals i no funcionals del sistema i presenta el disseny realitzat. Aquest inclou el model de base de dades, la interfície gràfica, la comunicació entre mòduls, l'estructura de carpetes i altres aspectes essencials per a la implementació.

En **Implementació** (4) es descriu en detall el procés de desenvolupament de l'aplicació. S'hi expliquen l'entorn de treball i les tecnologies emprades, així com la implementació de cadascun dels mòduls principals (*frontend*, *backend* de Node.js i *backend* de Python), les estratègies de seguretat adoptades i les dificultats trobades amb les seves respectives solucions.

En **Proves i Validació** (5) es presenten les proves realitzades per verificar el correcte funcionament del sistema. S'hi defineix l'estratègia de proves, es detallen els tests aplicats i es discuteixen els resultats obtinguts, juntament amb les limitacions detectades.

Finalment, en **Conclusions i Treball Futur** (6) es recull les conclusions generals del projecte, destacant les aportacions més rellevants, i proposa possibles millores i línies de treball futur que podrien ampliar el sistema desenvolupat.

## MARC TEÒRIC

### 2.1 Introducció al Marc Teòric

En aquest capítol es defineix el marc conceptual i tècnic necessari per comprendre el projecte. S'hi presenta, en primer lloc, el problema que es pretén resoldre, contextualitzant-lo dins del seu àmbit d'aplicació i destacant-ne la rellevància. A continuació, es descriu la idea inicial que va motivar el desenvolupament, així com les principals decisions preses durant les primeres fases de concepció.

També s'inclou una revisió de les tecnologies, eines i metodologies considerades, amb una justificació raonada de les escollides per dur a terme la solució.

### 2.2 Antecedents i Estat de l'Art

Els assistents conversacionals i els sistemes d'interacció amb bases de dades mitjançant llenguatge natural han experimentat un desenvolupament significatiu en els últims anys. Diversos projectes i investigacions han mostrat les possibilitats i limitacions d'aquest tipus de tecnologia, servint de base per al disseny i desenvolupament del nostre projecte.

Entre els projectes més rellevants es poden destacar:

- **IBM Watson Assistant** [1]: Plataforma que permet la creació d'assistents conversacionals capaços de processar llenguatge natural i integrar-se amb bases de dades i serveis externs. Malgrat la seva potència, el cost i la complexitat de configuració limiten l'adopció per a projectes petits o amb recursos reduïts.
- **Text-to-SQL amb OpenAI i LangChain** [2, 3, 4, 5]: Projectes recents han demostrat la viabilitat d'utilitzar models de llenguatge per traduir consultes en llenguatge

ge natural a sentències SQL. Aquests projectes mostren una millora considerable en la interacció amb bases de dades, reduint la necessitat de coneixements tècnics per part de l'usuari.

- **Xatbots locals amb llibreries com Ollama o LLaMA** [6, 7]: Permeten executar models de llenguatge en entorns locals, augmentant la privacitat i reduint la dependència d'Internet. No obstant això, el rendiment i la capacitat de gestió de consultes complexes són limitats respecte als models allotjats al núvol.

La combinació de LLMs allotjats al núvol amb integració directa amb bases de dades representa un avenç significatiu en comparació amb projectes previs. Els punts clau d'aquest avenç són:

- **Accessibilitat:** Els usuaris poden interactuar amb les dades sense necessitat de conèixer SQL ni altres llenguatges tècnics.
- **Escalabilitat i rendiment:** L'ús de models allotjats al núvol permet gestionar un volum alt de consultes simultànies sense afectar la qualitat de les respostes.
- **Flexibilitat i personalització:** L'elecció de models i la capacitat d'adjuntar context en fitxers millora la precisió de les respostes i permet adaptar l'assistent a necessitats concretes.
- **Reducció de costos i complexitat:** L'ús de models com GPT-4.1 mini ofereix un equilibri entre capacitat, velocitat i cost, facilitant la implementació en entorns empresarials amb recursos limitats.

Aquest estat de l'art mostra com la tecnologia ha evolucionat cap a assistents més intel·ligents, accessibles i eficients, justificant la selecció de les eines i models utilitzats en aquest projecte.

## 2.3 Conceptes Clau

Per comprendre plenament el projecte, és important establir una sèrie de conceptes fonamentals que seran recurrents al llarg del desenvolupament i la implementació. A continuació es defineixen els principals:

### 2.3.1 *Large Language Model* (Model de Llenguatge Gran) (LLM)

Els LLM són models de llenguatge de gran escala entrenats amb enormes quantitats de text per generar respostes coherents i contextualitzades. La seva arquitectura es basa sovint en xarxes neuronals, que permeten processar seqüències de paraules de manera eficient, capturant relacions a llarg termini dins del text. Els LLM poden executar tasques com traducció automàtica, generació de text, resposta a preguntes i, en aquest cas, traducció de llenguatge natural a consultes SQL.

### 2.3.2 Intel·ligència artificial, Aprenentatge Automàtic i Processament de Llenguatge Natural

- **Intel·ligència Artificial (IA)**: Disciplina de la informàtica que cerca desenvolupar sistemes capaços de dur a terme tasques que normalment requereixen intel·ligència humana.
- **Machine Learning (Aprenentatge Automàtic) (ML)**: Subcamp de la IA que se centra a crear models capaços d'aprendre de les dades i millorar el seu rendiment sense ser explícitament programats.
- **Natural Language Processing (Processament de Llenguatge Natural) (NLP)**: Àrea del ML i la IA que estudia com les màquines poden comprendre, interpretar i generar llenguatge humà.

## 2.4 Plantejament del Problema

El projecte neix dins el context d'una petita empresa formada per dos o tres treballadors que utilitzen diàriament el programari corporatiu. Els usuaris, tot i no estar familiaritzats amb les tecnologies més recents, coneixen bé l'estructura i el contingut de la base de dades de l'empresa. El l'equip informàtic està format per ordinadors de prestacions limitades, però es disposa d'un servidor local propi i d'un domini registrat, la qual cosa permet activar connexions segures mitjançant *Hypertext Transfer Protocol Secure* (Protocol de Transferència d'Hipertext Segur) (HTTPS) i a través de la red pública si fos necessari.

Actualment, el programari existent ofereix funcionalitats molt limitades, especialment pel que fa a la consulta i visualització de la informació emmagatzemada a la base de dades. Aquest procés resulta poc intuïtiu i requereix temps addicional de cerca i interpretació. El projecte proposa un sistema complet, centrant-se el marc d'aquest TFG en el desenvolupament d'un xatbot que actua com a assistent capaç d'agilitzar la consulta de dades i facilitar-ne la interpretació. Mitjançant l'ús de llenguatge natural, el bot permet reduir el temps dedicat a la visualització i raonament de les dades, alhora que millora la comprensió de la informació i pot oferir previsions sobre certs esdeveniments rellevants.

Per tal de garantir l'accessibilitat i minimitzar els requeriments tècnics, des d'un primer moment es considera que una aplicació web (Web App) és l'opció més adequada per al context plantejat. Aquest enfocament aporta les característiques següents:

- *Fàcil accés* des de qualsevol dispositiu: l'únic requisit és poder arribar a la IP on s'executa el contenidor *frontend* i disposar d'un navegador web modern que compleixi amb les especificacions necessàries.
- *Control d'accés* total als dispositius autoritzats: independentment del servei d'autenticació, és possible establir regles de Firewall o restringir l'accés a una xarxa local.

- *Integració senzilla* en el context empresarial: només cal un servidor amb recursos mínims (per exemple, 50 GB d'emmagatzematge són suficients en la majoria de casos) i la infraestructura es pot engegar o aturar segons les necessitats.
- *Actualització centralitzada*: totes les millores i correccions es despleguen al servidor, garantint que tots els usuaris utilitzin la versió més recent sense instal·lacions locals.
- *Compatibilitat multiplataforma*: funciona de manera consistent a Windows<sup>1</sup>, Linux<sup>2</sup>, macOS<sup>3</sup>, Android<sup>4</sup> i iOS<sup>5</sup> sense adaptar codi específic per a cada entorn, fet que permet l'ús en dispositius antics.
- *Facilitat d'integració amb altres serveis*: integració àgil amb APIs, serveis interns o bases de dades mitjançant protocols web estàndard.
- *Seguretat centralitzada*: la gestió d'autenticació, permisos i registre d'activitat es controla des del servidor, assegurant que només els usuaris autoritzats puguin utilitzar el bot i accedir a les dades.

Els requisits principals del sistema inclouen la disponibilitat durant l'horari d'oficina (amb la possibilitat d'operar 24/7 si es detecta demanda fora d'aquest horari), compatibilitat amb una àmplia varietat de dispositius, seguretat completa en el tractament de dades, una interfície de xat intuïtiva i un disseny visual coherent amb la identitat corporativa.

Des de la seva concepció, es planteja un *disseny modular* per garantir la independència de cada component. Aquest enfocament incrementa la resiliència davant possibles errors i facilita una programació escalable i horitzontal. Així, per exemple, si la base de dades deixa de funcionar, l'usuari pot continuar interactuant amb el *frontend* i amb certes funcionalitats del *backend*, i viceversa.

Per a una fàcil instal·lació el projecte serà de codi obert i es podrà descarregar desde internet, a més, la instal·lació mínima només requereix l'eina Docker i una petita configuració de certes variables a tenir en compte.

### 2.5 Tecnologies i Eines

En aquesta secció es comenten les eines usades durant el disseny i desenvolupament del projecte, explicant el perquè d'aquestes i en alguns casos les alternatives. En general, totes aquestes tecnologies comparteixen el fet que ja es tenia un coneixement previ sobre aquestes o que són tecnologies molt usades amb una comunitat molt gran.

---

<sup>1</sup> Sistema operatiu usat per la majoria d'usuaris i desenvolupat per l'empresa Microsoft.

<sup>2</sup> Sistema operatiu de codi obert.

<sup>3</sup> Sistema operatiu exclusiu d'ordinadors desenvolupat per l'empresa Apple.

<sup>4</sup> Sistema operatiu basat en Linux i pensat principalment per dispositius mòbils.

<sup>5</sup> Sistema operatiu desenvolupat per Apple que es centra en ser el sistema per als *iPhones*.



### 2.5.1 Entorn de Desenvolupament

Per garantir la coherència del procés de desenvolupament i la replicabilitat dels resultats, era essencial comptar amb un entorn estable i portable, capaç de funcionar de manera consistent en diferents sistemes operatius i configuracions.

Durant el desenvolupament de l'aplicació es va emprar únicament l'entorn Debian. Les diferents eines utilitzades es van integrar dins l'editor de codi VSCode. Tot i així, l'ús inicial de contenidors i les proves finals van permetre verificar el funcionament correcte tant en Linux com en Windows i en la majoria de dispositius.

Per dur un control dels canvis del projecte s'ha emprat Git i per guardar tots els arxius i la seva futura publicació s'empra [GitHub](#).

Per als contenidors de cada mòdul es va decidir l'eina Docker, ja que és la més estandaritzada pel seu bon funcionament i per ser Open Source (Codi Obert).

Per al disseny inicial de la base de dades s'ha emprat la web [drawDB](#) que ha ajudat en l'adaptació de les taules ja existents, la creació de noves i les relacions entre aquestes.

### 2.5.2 Tecnologies i Llenguatges

La selecció de tecnologies i llenguatges va buscar un equilibri entre l'experiència prèvia del desenvolupador, la popularitat de les eines i el suport de la comunitat, assegurant així una implementació eficient i mantenible del projecte.

Quant al conjunt de llenguatges i tecnologies més específiques per a cada funció:

1. *Base de dades SQL amb el sistema PostgreSQL*: es va optar per un model relacional atès que l'estructura de les dades és altament estable i es beneficia d'un esquema rígid. El Sistema Gestor de Bases de Dades (SGBD) PostgreSQL es va seleccionar per la seva robustesa, escalabilitat i naturalesa Open Source (Codi Obert), així com pel suport nadiu a tipus de dades avançats i funcions procedurals. Alternatives viables haurien estat *MySQL*, *SQLite* o *MariaDB*.
2. *Base de dades NoSQL Redis per a memòria cau*: es va incorporar per optimitzar la latència en operacions freqüents mitjançant l'emmagatzematge en memòria volàtil. La seva arquitectura clau-valor, l'eficiència en la gestió de dades transitories i el baix consum de recursos, juntament amb el seu caràcter Open Source (Codi Obert), la converteixen en un estàndard de facto en sistemes d'alt rendiment.
3. *Vite*: s'ha utilitzat com a servidor de desenvolupament per la seva capacitat de compilació i actualització incremental, reduint el temps de *build* i permetent una visualització quasi immediata dels canvis en el codi. La seva arquitectura minimitza la sobrecàrrega de configuració inicial.
4. *React framework* de JavaScript escollit per la seva arquitectura basada en components declaratius i el seu Virtual DOM, que optimitza el renderitzat. Es va valorar l'ecosistema madur i la corba d'aprenentatge moderada. Alternatives com

*Angular* o *Vue* també eren candidates, però amb requeriments d'aprenentatge o filosofies de disseny diferents.

5. *Tailwind CSS: framework* de *Cascading Style Sheets* (CSS) basat en utilitats que permet estandarditzar estils i reduir la dependència de fulls CSS externs. L'enfocament d'atributs dins del marcatge *Hypertext Markup Language* (Llenguatge de Marcatge d'Hipertext) (HTML) accelera el desenvolupament i garanteix consistència visual.
6. *Node.js*: entorn d'execució triat per al *backend* amb l'objectiu de reutilitzar el mateix llenguatge que al *frontend*, minimitzant la corba d'aprenentatge i reduint la complexitat de l'entorn de desenvolupament. El suport a l'Express *framework*, la gestió eficient d'I/O no bloquejant i l'ecosistema extens de paquets NPM van ser factors determinants.<sup>6</sup>
7. *JSON Web Token (JWT)*: mecanisme d'autenticació i autorització que utilitza *tokens* auto-contenidors signats digitalment. La seva adopció es va justificar per l'eliminació de la necessitat d'emmagatzematge en servidor, la compatibilitat amb entorns distribuïts i la seva àmplia acceptació en aplicacions web modernes.
8. *Python*: llenguatge seleccionat per a mòduls específics del *backend*, especialment per la seva preeminència en l'àmbit de la IA i ML, així com per l'extensa disponibilitat de llibreries per a la integració amb models LLM.
9. *FastAPI: framework* adoptat per la creació d'APIs amb Python gràcies a la seva arquitectura asíncrona, documentació automàtica amb *OpenAPI* i sintaxi tipada. Va ser preferit davant alternatives com *Flask* per la seva major eficiència i integració nativa amb mecanismes de validació de dades.

### 2.5.3 Models de Llenguatge i Selecció de l'Assistent

Els models de llenguatge constitueixen la peça central de la solució, ja que determinen la capacitat de l'assistent per comprendre el llenguatge natural, generar respostes adequades i interactuar amb la base de dades de manera precisa i eficaç

Inicialment es va provar amb la llibreria *Ollama*, que permet executar models LLM en local, la qual oferia l'avantatge de no requerir connexió a Internet i augmentar la seguretat. Tot i així, els models provats (*mistral*, *llama3*) no van oferir bons resultats en consultes complexes a la BD, fet que va motivar la transició cap a models allotjats al núvol d'OpenAI, decidint prescindir de models locals a canvi de millorar la potència i rendiment del bot.

Aquí és on es comencen a fer proves usant la llibreria de OpenAI i a usar els diferents models oferits. Tots els tests amb la majoria dels models foren molt satisfactoris així que es va decidir seguir el desenvolupament amb aquesta eina.

A causa de la gran quantitat d'opcions oferides com a models conversacionals es va decidir generalitzar i donar més llibertat a l'usuari podent elegir el seu model gràcies a

---

<sup>6</sup>Input/Output (o Entrada/Sortida) fa referència a les operacions on una aplicació llegeix o envia dades fora de la CPU.

una variable d'entorn del projecte. Així i tot, per decidir elegir el model predeterminat per ser l'assistent conversacional de l'empresa Vismel es va fer un estudi comparatiu de rendiment, funcionalitats, cost i limitacions de les opcions més atractives<sup>7</sup>.

Els models d'OpenAI tenen diferents especialitats que s'enfoquen en certes funcions, especialment en dos àmbits rellevants per al nostre projecte:

- **Intel·ligència general:** Models dissenyats per mantenir converses fluides amb usuaris, amb gran comprensió del llenguatge natural. Són adequats per tasques conversacionals i de comprensió contextual sense un enfocament tècnic específic. Exemples: família GPT.
- **Raonament:** Models optimitzats per resoldre problemes complexos, generar codi o realitzar càlculs avançats. Dedicar més temps al procés de planificació i resolució, però amb menor fluïdesa en la conversa. Exemples: família o3.

Pel cas del nostre assistent, es va prioritzar la *intel·ligència general*, ja que la seva tasca principal serà mantenir converses i generar sentències SQL senzilles, sense necessitat de raonament profund.

Les característiques necessàries del xatbot són:

- Prou Capacitat per respondre correctament a la majoria de les consultes dels usuaris.
- Velocitat de resposta raonable per assegurar una bona UX i assegurar uns requisits no funcionals mínims.
- Per a consultes on el context sigui molt llarg el model ha de suportar la capacitat d'adjuntar fitxers que contindran el context complet.
- Un preu raonable per l'ús que es dona i les cridades a la API de OpenAI.
- Un límit de tokens prou gran per a poder adjuntar en la majoria dels casos el context complet sense la necessitat d'adjuntar fitxers i així millorar els temps de resposta.

S'han seleccionat els 6 models més usats en aquest àmbit i es farà una comparativa per seleccionar un com el predeterminat:

[8]<sup>8,9</sup>

---

<sup>7</sup>A l'hora de desenvolupar el projecte l'empresa OpenAI encara no havia presentat els seus nous models que l'arquitectura base *GPT-5* per tant, no apareixen als estudis comparatius.

<sup>8</sup>El preu mitjà dels models és entre entrades, entrades de *cache* i sortides, tot per cada 1 milió de tokens. Aproximadament 4 caràcters equivalen a 1 token.

<sup>9</sup>L'*endpoint* de *v1/assistants* és necessari en aquest projecte per la forma en què es tracten els fitxers a les consultes.

Aspecte	GPT-3.5 Turbo	GPT-4	GPT-4o	GPT-4o mini	GPT-4.1	GPT-4.1 mini
Intel·ligència	1	2	3	2	4	3
Velocitat	2	3	3	4	3	4
Permet v1/assistants	No	Sí	Sí	Sí	Sí	Sí
Preu Mitjà	1 €	45 €	4,58 €	0,27 €	3,5 €	0,7 €
Límit de Tokens	16.385	8.192	128.000	128.000	1.047.576	1.047.576
TPM	200.000	10.000	30.000	200.000	30.000	200.000

Taula 2.1: Comparació Models OpenAI

Com veiem a la taula el model 3.5 ja es pot descartar perquè no suporta l'adjunció de fitxers. Dels 5 restants el GPT-4 és superat en tots els punts per algun dels altres models, per tant, també és descartat. Per la resta tots tenen unes característiques més o manco raonables i suficients, en conseqüència, s'han dut a terme proves de recerca i test dels 4 models.

Per a consultes simples la diferència no és gaire notòria, mentre que per consultes més complexes on es necessiten extreure moltes dades de la BD o generar *queries* més sofisticades tots responen correctament i de manera coherent, però els models GPT-4o i GPT-4.1 bàsics donen unes respostes més ben raonades i s'apropen més al pensament humà.

A la taula podem veure que els models GPT-4.1 superen sobre el paper en velocitat, intel·ligència i límit de tokens als GPT-4o. A més s'ha realitzat una recerca per saber les especialitats de cada família i quina pot encaixar dins el context del xatbot on clarament el guanyador és la família GPT-4.1.

1. Com es veu a la taula 2.1 la diferència més notòria és el límit de tokens al context. Això ja és un benefici per la família GPT-4.1, ja que ens permet afegir més dades directament al missatge fora fer ús de l'adjunció d'arxius i millorant els temps de resposta.
2. Veiem que segons OpenAI les velocitats són similars, però hi ha articles que comenten com GPT-4.1 sol respondre de manera més ràpida [9].
3. Per tasques específiques GPT-4o es va programar com un model LLM multimodal, és a dir, que pot processar text, imatges, àudio i vídeo. Per altra banda, GPT-4.1 està optimitzat per tasques de programació, instruccions complexes i grans contextos. En aquest cas el xatbot de Vismel només processa text (de moment) i tenint en compte que la majoria dels pics el model haurà de generar codi SQL llavors la família GPT-4.1 pot tenir un millor rendiment [10, 11].
4. En termes de rendiment programant es mostra com GPT-4.1 té una millora notable davant GPT-4o. En SWE-Bench GPT-4.1 té una precisió d'aproximadament 55% mentre que GPT-4o només se situa en un 33% [10, 12, 13, 14]. També els models 4.1 mostren més fiabilitat generant codi nou que es traspasa a la capacitat de seguir i respectar instruccions complexes i extenses, cosa que és molt

important, ja que, com es podrà veure en capítols posteriors, el xatbot funciona a partir d'aquestes [10, 15].

Per tant, la qüestió final es dona entre el model 4.1 i el 4.1 mini. Finalment, es va decidir pel model GPT-4.1 bàsic perquè com hem comentat abans, encara que sigui una mica més car, dona respostes més ben raonades i més properes al llenguatge humà i el preu arriba a ser irrisori per l'ús real que es preveu a l'empresa amb només 2 o 3 persones usant-ho.

## 2.6 Arquitectura del Sistema

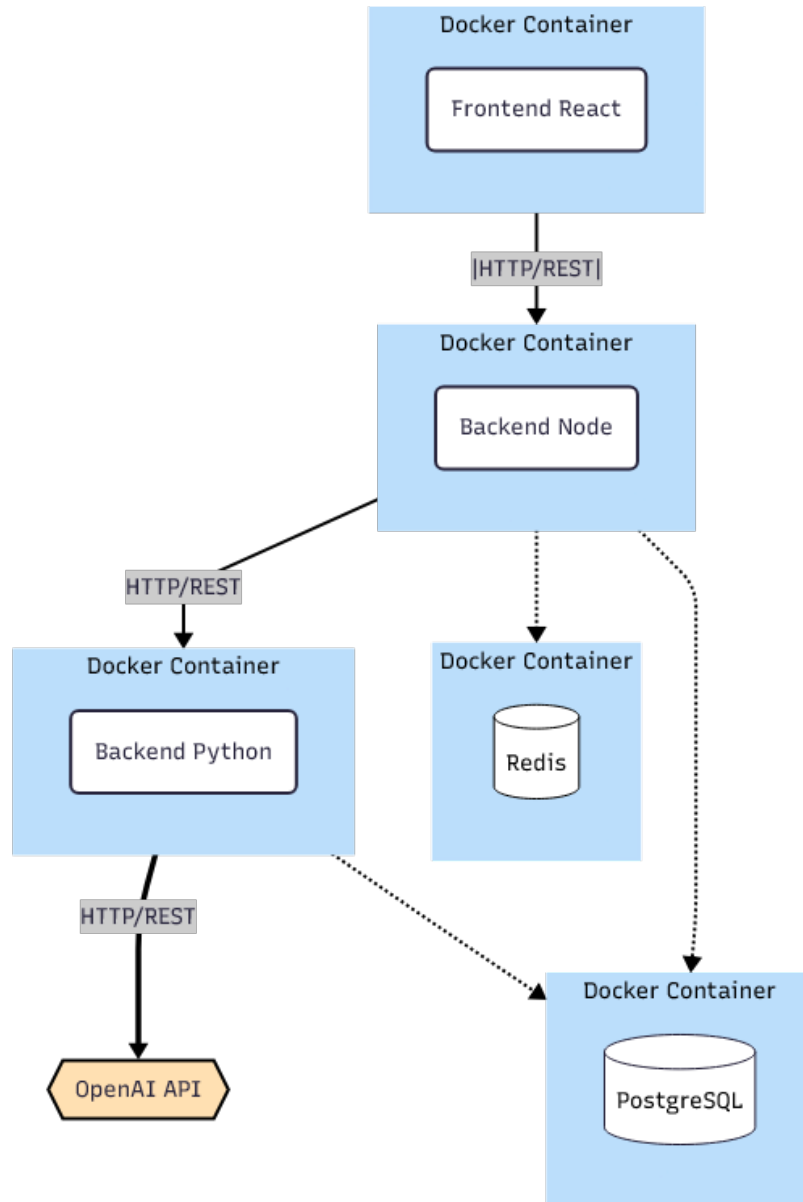


Figura 2.1: Diagrama de fluxe dels mòduls del projecte

Cada component del projecte s'executa dins del seu propi contenidor Docker. Això garanteix l'aïllament de processos, però permet la comunicació entre ells quan és necessari.

El flux de funcionament és el següent:

1. L'usuari accedeix al *frontend* des del navegador, inicia sessió i formula una con-

sulta al xatbot.

2. Les peticions, tant d'autenticació com de conversa, arriben al *backend* principal (Node.js) mitjançant peticions *Hypertext Transfer Protocol* (Protocol de Transferrència d'Hipertext) (HTTP).
3. Segons la naturalesa de la petició, el *backend* de Node.js pot:
  - Enviar la consulta al *backend* de Python perquè la processi el bot.
  - Consultar, inserir o modificar dades a la base de dades principal (PostgreSQL).
  - Accedir a Redis per llegir o escriure dades en memòria *cache*.
4. Quan el *backend* de Python rep una consulta procedent de Node.js, pot iniciar un o més intercanvis amb la API d'OpenAI. Durant aquest procés, si és necessari, també pot realitzar consultes a la base de dades de PostgreSQL.
5. Finalment:
  - Si es tracta d'una resposta generada pel bot, Python la retorna a Node.js, que l'envia al *frontend* per mostrar-la a l'usuari.
  - Si és una petició gestionada directament per Node.js (com la validació de credencials), la resposta es retorna immediatament al *frontend*.

## 2.7 Seguretat i Privacitat

Donat que el xatbot empresarial té accés a informació interna i potencialment sensible de l'empresa, la seguretat i la privacitat són pilars fonamentals en el seu disseny i implementació. El sistema s'ha construït amb un conjunt de mesures que garanteixen que només els usuaris autoritzats puguin accedir a les funcionalitats i dades corresponents al seu rol, alhora que es protegeix la informació durant tot el cicle de vida de la consulta.

L'aplicació implementa un sistema d'autenticació basat en JWT:

- Quan l'usuari inicia sessió amb les seves credencials, el servidor valida la informació i genera un JWT signat amb una clau secreta del *backend*.
- Aquest token s'envia al client i s'adjunta automàticament a cada petició posterior, cosa que permet que el servidor verifiqui la identitat de l'usuari sense haver de demanar credencials contínuament.
- El sistema d'autorització assigna rols (p. ex., administrador, empleat, programador) i verifica, per cada petició, que el rol tingui permisos per executar l'acció sol·licitada.

Mesures de protecció de dades:

- **Xifratge en trànsit:** Totes les comunicacions entre client i servidor es realitzen sobre HTTPS.
- **Control d'accessos:** La BD PostgreSQL està protegida amb credencials robustes i configuracions de *firewall* per restringir accessos externs.
- **Redis:** Utilitzada únicament com a memòria cau per dades temporals no crítiques. Pot contenir informació sensible de l'històric de converses, però l'accés a la BD i a les dades a través de l'API de Node.js requereix un usuari autoritzat.

Consideracions de privacitat:

- Evitar emmagatzemar dades de clients que no siguin estrictament necessàries per al funcionament de l'aplicació i prèviament autoritzades.
- En l'ús de l'API d'OpenAI, només s'envien dades estrictament requerides per respondre la consulta.
- Les dades més sensibles com contrasenyes es xifren abans de guardar a la BD.

En conjunt, aquestes mesures garanteixen que el xatbot pugui operar de forma segura en un entorn empresarial. Els riscos són limitats, ja que l'aplicació està pensada per executar-se localment i no exposar-se a tràfic exterior, tot i que aquestes mesures ofereixen una capa addicional de seguretat. A més assegura que només els usuaris amb més autoritat de l'oficina puguin accedir a les dades.



## DISSENY I REQUISITS

### 3.1 Introducció al Disseny i Requisits del Projecte

En aquest capítol es descriuran els requisits del projecte, tant funcionals com no funcionals, establerts per l'empresa i per l'autor, amb l'objectiu de garantir el compliment d'uns estàndards de qualitat i coherència.

Aquesta anàlisi preliminar dels requisits permetrà comprendre amb major profunditat i claredat el disseny de cada mòdul, que s'explicaran d'una manera més detallada i tècnica.

### 3.2 Requisits del Sistema

La principal funcionalitat del projecte, un xatbot amb IA integrat amb la base de dades interna, ha d'estar dissenyat per respondre consultes complexes de forma senzilla, accessible i segura.

La definició dels requisits és essencial per garantir que el sistema compleixi les expectatives dels usuaris, assegurant que l'eina sigui útil, intuïtiva i fiable.

#### 3.2.1 Requisits Funcionals

Els requisits funcionals són aquells que defineixen les capacitats i comportaments del projecte. Així, es poden definir els següents:

1. Els usuaris han de poder accedir a l'aplicació en qualsevol moment a través del navegador *Web*.
2. L'usuari ha de ser capaç d'iniciar sessió dins l'aplicació amb un nom d'usuari i una contrasenya.

3. L'usuari també ha de ser capaç de tancar la sessió sempre que vulgui.
4. L'usuari ha de tenir un rol assignat que definirà les funcionalitats que pot usar i les dades que pot veure dins l'aplicació.
5. Els usuaris administratius han de poder veure tots els usuaris existents de l'aplicació i tenir la capacitat d'afegir, modificar o eliminar qualsevol d'ells.
6. Els usuaris administratius han de poder veure una pàgina amb una sèrie de gràfiques que mostren un resum de les dades més importants.
7. Qualsevol usuari pot veure (només el seu) l'historial de missatges complet que ha tingut en qualsevol moment.
8. Si així ho decideix, l'usuari ha de poder començar un nou xat esborrant els missatges anteriors.
9. Si l'usuari té els permisos corresponents ha de ser capaç d'escriure una consulta a un bot i aquest ha de poder respondre correctament.
10. El bot és conscient en tot moment del seu rol com a assistent empresarial i ha de tenir un context complet i ben estructurat per donar bones respostes.
11. L'assistent ha de poder extreure informació de la BD per poder millorar la resposta a la consulta de l'usuari.

#### 3.2.2 Requisits No Funcionals

Els requisits no funcionals tracten de característiques de qualitat, rendiment i seguretat sense incidir en les funcionalitats concretes del projecte. Aquests són els que s'han definit segons les necessitats dels usuaris de l'oficina de Vismel i altres raonats per l'autor i el tutor del TFG:

1. Les dades han d'estar segures en tot moment dins la BD sent només els usuaris autoritzats els únics que les poden veure.
2. Les dades més sensibles com contrasenyes han d'estar encriptades per assegurar les bones pràctiques de protecció de dades.
3. El xatbot ha de poder respondre el més ràpid possible. Per a les consultes senzilles ha de tenir una resposta en menys de 3 segons. Les funcionalitats han de tenir una resposta ràpida, per tant, l'extracció de dades ha de ser el més ràpid possible usant *cache* i consultes SQL simplifiades.
4. El xatbot ha de generar sempre una resposta encara que hi hagi alguna falla o no pugui accedir a les dades però sense inventar res.
5. El sistema ha de ser resilient a falles. Si un mòdul no funciona l'aplicació no ha de deixar de funcionar.

6. El sistema ha d'estar disponible sempre que sigui possible. Això s'aconsegueix amb un reinici automàtic dels contenidors.
7. La interfície ha de ser senzilla i intuïtiva per assegurar la usabilitat de l'aplicació.
8. Tots els usuaris han de comprendre les diferents parts de l'aplicació i les respostes del bot (sempre respondrà en espanyol).
9. L'aplicació ha de seguir una estètica similar a la de l'empresa (pocs colors, molt bàsics i els principals en escala de grisos).
10. La bona funcionalitat i simplicitat ha de tenir prioritat davant la UX/UI. No deixa de ser una eina de treball.
11. Arquitectura i estructura del projecte escalables pensat per créixer l'aplicació en poc temps, modularitzant els serveis i treballant amb bones tècniques de programació com Programació Orientada a Objectes (POO).
12. Independència del Sistema Operatiu (SO) on corre i dels dispositius que accedeixen a l'aplicació, això s'aconsegueix amb els contenidors de Docker que poden córrer en la majoria de SOs.

### 3.3 Disseny de la Base de Dades

Per assegurar que el bot pot obtenir correctament les dades i de la forma més eficient assegurant el millor rendiment possible es necessitava una actualització de la BD que l'empresa havia usat fins ara tant per la tecnologia com per l'estructura d'aquesta, per això es van aplicar certes tècniques per crear el millor disseny possible de la nova BD.

A més es necessitava una memòria on guardar dades que havien de sobreviure entre sessions de navegador, però no eren del tot persistents, per això es va optar per usar la *cache*.

#### 3.3.1 Base de Dades PostgreSQL

El primer pas implica tenir un bon disseny de la BD principal i la millor manera era estudiar com es treballava dins l'empresa i com eren gestionades les dades fins ara. Aquestes es trobaven dins una BD molt antiquada i amb nul manteniment, on es veien taules de mides desproporcionades, mesclant dades de diferents tipus i fora correlació entre elles.

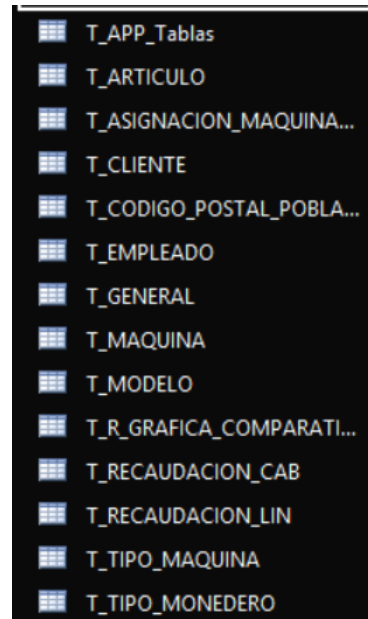


Figura 3.1: Exemple taules de la Base de Dades que usava Vismel

Després de revisar amb l'usuari administrador les taules amb les dades realment usades i que tenien més importància es va poder definir els aspectes principals que l'empresa necessitava:

- L'aspecte central són les màquines expenedores, amb totes les seves característiques pròpies.
- Les màquines només tenen un client associat. Els clients no s'identifiquen per l'empresa o entitat sinó pel lloc on se situen i aquests poden tenir més d'una màquina. També tenen dades pròpies d'interès.
- El client sempre estarà en una població identificada per un codi postal.
- A l'empresa hi ha diferents empleats amb diferents nivells de responsabilitat. Cada empleat pot tenir associades certes màquines i tasques com revisions i avaries.
- L'empresa té un registre setmanal de cada màquina amb la recaptació. D'aquesta recaptació s'encarrega un empleat concret.
- Hi ha un inventari de diferents articles que van entrant i sortint del magatzem. Sempre hi ha un empleat associat a l'acció de ingressar o extreure un article.

Tenint en compte aquesta base sobre la qual se sustenta l'empresa es varen crear diferents taules i relacions entre elles ben optimitzades i fora ambigüitats donant com a resultat la següent BD:

### 3.3. Disseny de la Base de Dades

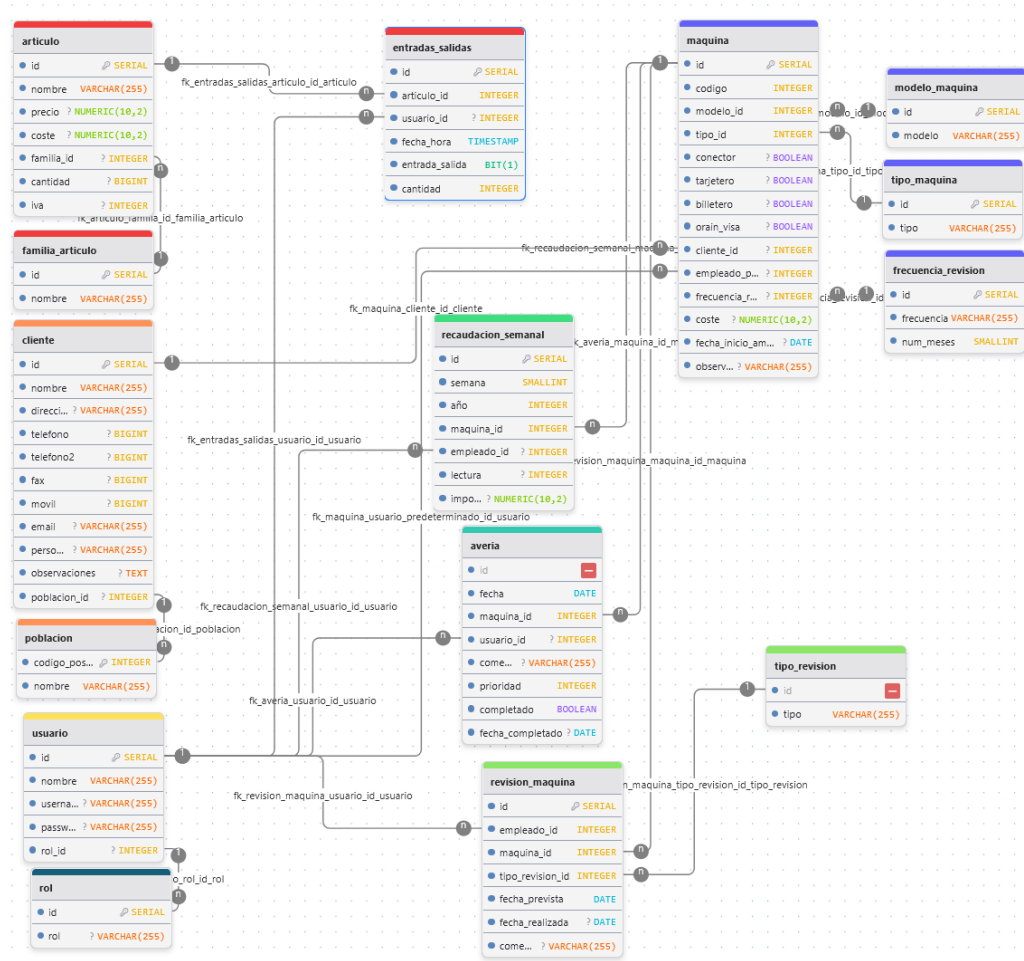


Figura 3.2: Diagrama de la Base de Dades

#### 1. Taula usuario

La taula usuario emmagatzema informació sobre els empleats que treballen a l'empresa i que interactuen amb les màquines, els clients i l'inventari. Cada usuari pot estar associat a revisions de màquines, registre d'avaries, moviments d'inventari i recaptacions. El rol de l'usuari defineix quines accions pot fer dins del sistema.

Columna	Descripció
id	Identificador únic de cada usuari.
nombre	Nom complet de l'usuari.
username	Nom d'usuari para accedir al sistema.
password	Contrasenya de l'usuari par l'autenticació.
rol_id	Referencia al rol assignat, determinant els seus privilegis.

Taula 3.1: Columnes de la taula usuario

#### 2. Taula rol

### 3. DISSENY I REQUISITS

---

La taula `rol` classifica els diferents nivells de privilegis que poden tenir els empleats.

Columna	Descripció
<code>id</code>	Identificador únic del rol.
<code>rol</code>	Nom o descripció del rol (per exemple, administrador).

Taula 3.2: Columnes de la taula `rol`

#### 3. Taula `maquina`

La taula `maquina` representa cada màquina expenedora gestionada per l'empresa, incloent-hi les característiques tècniques, l'empleat associat i el manteniment. Cada màquina s'associa a un client, un model i un tipus. A més, vincula revisions, avaries i recaptacions, funcionant com a element central de la base de dades.

Columna	Descripció
<code>id</code>	Identificador únic de la màquina.
<code>codigo</code>	Codi intern de la màquina.
<code>modelo_id</code>	Referència al model de la màquina.
<code>tipo_id</code>	Referència al tipus de màquina.
<code>connector</code> , <code>targeter</code> , <code>bitlleter</code> , <code>orain_visa</code>	Indiquen funcions o components addicionals de la màquina.
<code>cliente_id</code>	Referència al client on està ubicada la màquina.
<code>empleado_predeterminado_id</code>	Empleat responsable per defecte de la màquina.
<code>frecuencia_revision_id</code>	Indica cada quant s'ha de revisar la màquina.
<code>coste</code>	Cost d'adquisició de la màquina.
<code>fecha_inicio_amortizacion</code>	Data d'inici del càlcul d'amortització.
<code>observaciones</code>	Notes addicionals sobre la màquina.

Taula 3.3: Columnes de la taula `maquina`

#### 4. Taula `tipo_maquina`

La taula `tipo_maquina` categoritza les màquines segons la funció o el disseny. Cada màquina està associada a un tipus.

Columna	Descripció
<code>id</code>	Identificador únic del tipus.
<code>tipo</code>	Descripció del tipus de màquina.

Taula 3.4: Columnes de la taula `tipo_maquina`

#### 5. Taula `modelo_maquina`

La taula `modelo_maquina` identifica els models específics de les màquines. Permet vincular cada màquina amb el seu model.

Columna	Descripció
<code>id</code>	Identificador únic del model.
<code>modelo</code>	Nom o codi del model.

Taula 3.5: Columnes de la taula `modelo_maquina`

#### 6. Taula frecuencia\_revision

Defineix cada quant de temps s'ha de fer la revisió d'una màquina. Cada màquina té associada una freqüència de revisió que serveix per planificar manteniments preventius.

Columna	Descripció
id	Identificador únic de la freqüència.
frecuencia	Descripció de la periodicitat (per exemple, mensual, trimestral).
num_meses	Nombre de mesos entre revisions.

Taula 3.6: Columnes de la taula frecuencia\_revision

#### 7. Taula revision\_maquina

Registra totes les revisions fetes a les màquines. Connecta màquines, empleats i tipus de revisió, permetent portar un historial complet de manteniments.

Columna	Descripció
id	Identificador únic de la revisió.
empleado_id	Referència a l'usuari que fa la revisió.
maquina_id	Referència a la màquina revisada.
tipo_revision_id	Tipus de revisió realitzada.
fecha_prevista	Data programada de la revisió.
fecha_realizada	Data en què realment es va efectuar la revisió.
comentarios	Observacions sobre la revisió efectuada.

Taula 3.7: Columnes de la taula revision\_maquina

#### 8. Taula tipo\_revision

Defineix els diferents tipus de revisions que es poden fer a les màquines. Permet classificar les revisions registrades a revision\_maquina, facilitant la gestió i anàlisi de manteniments.

Columna	Descripció
id	Identificador únic del tipus de revisió.
tipo	Descripció del tipus de revisió.

Taula 3.8: Columnes de la taula tipo\_revision

#### 9. Taula recaudacion\_semanal

Registra la recaptació de cada màquina de manera setmanal. Permet portar un control econòmic de les màquines i de la caixa feta.

### 3. DISSENY I REQUISITS

---

Columna	Descripció
id	Identificador únic del registre.
semana	Número de la setmana.
año	Any de la recaptació.
maquina_id	Referència a la màquina recaptada.
empleado_id	Usuari encarregat de la recaptació.
lectura	Valor registrat de la recaptació.
importe_recaudado	Suma total recaptada a la setmana.

Taula 3.9: Columnes de la taula `recaudacion_semana1`

#### 10. Taula `averia`

Controla les incidències i fallades de les màquines. Permet planificar i controlar la resolució d'incidències a les màquines, vinculant empleats i recursos necessaris.

Columna	Descripció
id	Identificador únic de l'avaria.
fecha	Data de registre de l'avaria.
maquina_id	Màquina afectada.
usuario_id	Empleat que registra o atén l'avaria.
comentarios	Descripció de la incidència.
prioridad	Nivell d'urgència de l'avaria.
completado	Indica si l'avaria va ser solucionada.
fecha_completado	Data de resolució de l'avaria.

Taula 3.10: Columnes de la taula `averia`

#### 11. Taula `cliente`

Emmagatzema informació sobre els clients on s'ubiquen les màquines. Cada client pot tenir diverses màquines assignades.

Columna	Descripció
id	Identificador únic del client.
nombre	Nom o denominació del client.
direccion	Adreça física del client.
telefono, telefono2, fax, movil	Dades de contacte.
email	Correu electrònic del client.
persona_contacto	Nom del contacte principal.
observaciones	Informació addicional rellevant.
poblacion_id	Referència a la població on es troba el client.

Taula 3.11: Columnes de la taula `cliente`

#### 12. Taula `poblacion`

Defineix les poblacions geogràfiques dels clients. Facilita la classificació de clients i màquines per ubicació geogràfica, útil per a logística i planificació de rutes de manteniment.



Columna	Descripció
codigo_postal	Codi únic de la població.
nombre	Nom de la població.

Taula 3.12: Columnes de la taula poblacion

**13. Taula articulo**

Registra els articles gestionats al magatzem. Permet controlar l'inventari i vincular moviments d'entrada i de sortida de productes amb empleats responsables.

Columna	Descripció
id	Identificador únic de l'article.
nombre	Nom del producte.
precio	Preu de venda.
coste	Cost d'adquisició.
familia_id	Referència a la família de l'article.
cantidad	Unitats disponibles.
iva	Percentatge d'IVA aplicat.

Taula 3.13: Columnes de la taula articulo

**14. Taula familia\_articulo**

Classifica els articles en grups. Facilita l'organització de l'inventari i l'anàlisi per categories de productes.

Columna	Descripció
id	Identificador únic de la família.
nombre	Nom de la família d'articles.

Taula 3.14: Columnes de la taula familia\_articulo

**15. Taula entradas\_salidas**

Registra tots els moviments d'inventari. Permet la traçabilitat dels articles, controlant l'estoc i vinculant moviments amb els responsables.

Columna	Descripció
id	Identificador únic del registre.
articulo_id	Article que entra o surt.
usuario_id	Empleat que realitza el moviment.
fecha_hora	Data i hora del registre.
entrada_salida	Indica si l'article entra o surt del magatzem.
cantidad	Nombre d'unitats mogudes.

Taula 3.15: Columnes de la taula entradas\_salidas

**3.3.2 Base de Dades Redis**

Redis és una base de dades en memòria de tipus clau-valor, és a dir, cada registre es guarda com un parell clau-valor, i el valor pot ser diversos tipus de dades: strings,

l·listes, diccionaris, etc. Guarda les dades en memòria *Random Access Memory* (Memòria d'Accés Aleatori) (RAM), cosa que permet accessos molt ràpids. Té funcions molt útils com establir un temps de vida per a una clau (EXPIRE) o que treballen directament amb l·listes (LPUSH i R PUSH).

En el cas del TFG l'únic ús donat a la memòria cau ha estat desfer l'historial de xat de l'usuari. Cada usuari té una entrada on seu id és la clau i el valor és una llista on cada element és un objecte amb dues claus, la clau *user* amb el missatge de l'usuari i la clau *bot* amb la resposta de l'assistent.

## 3.4 Interfície Gràfica

S'ha dissenyat una Interfície Gràfica (IG) bàsica que supleix totes les necessitats de l'aplicació i que dona una idea de la ruta a seguir per a la seva versió final.

La primera pàgina que ens trobem és la d'inici de sessió (3.3). Aquí els usuaris s'han de registrar amb les seves credencials (nom d'usuari i contrasenya) per poder accedir a la resta de funcions de l'aplicació. Això es fa per assegurar la identitat dels usuaris i determinar el seu rol, ja que no tots tenen el mateix accés a les funcionalitats i a les dades. Veiem que no es poden registrar perquè serà l'administrador des de l'aplicació l'encarregat de crear els usuaris que poden accedir.

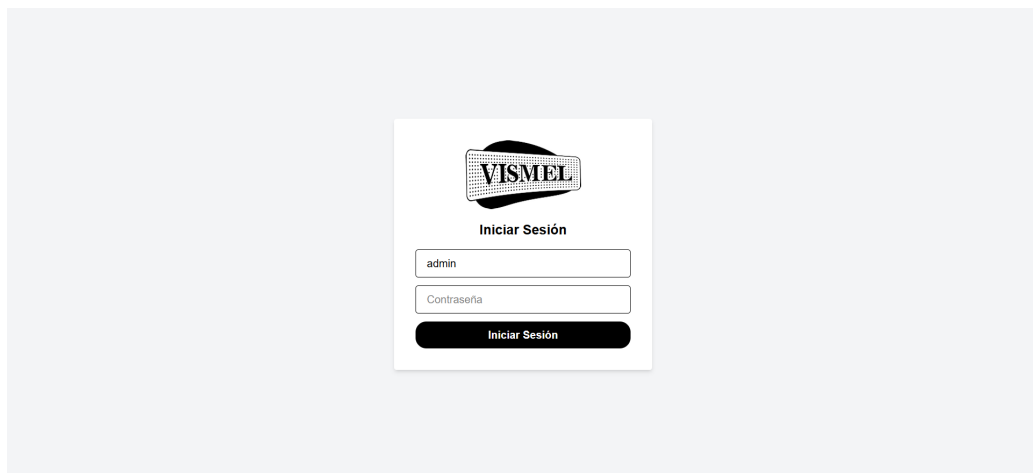


Figura 3.3: Pàgina d'inici de sessió

A l'iniciar sessió, independentment del rol de l'usuari, es mostra la pàgina *home* (3.4) que és simplement una pàgina estàndard amb el logo de l'empresa i el nom de l'usuari. L'usuari té una barra de navegació amb les funcions a les quals pot accedir, també a la dreta d'aquesta barra es mostra informació de l'usuari i un desplegable amb altres funcions. Actualment, només hi ha funcions disponibles pels usuaris administradors, per tant, qualsevol altre tipus d'usuari no tindrà cap pàgina que visitar menys la de *home* i la funció de tancar la sessió.

### 3.4. Interfície Gràfica

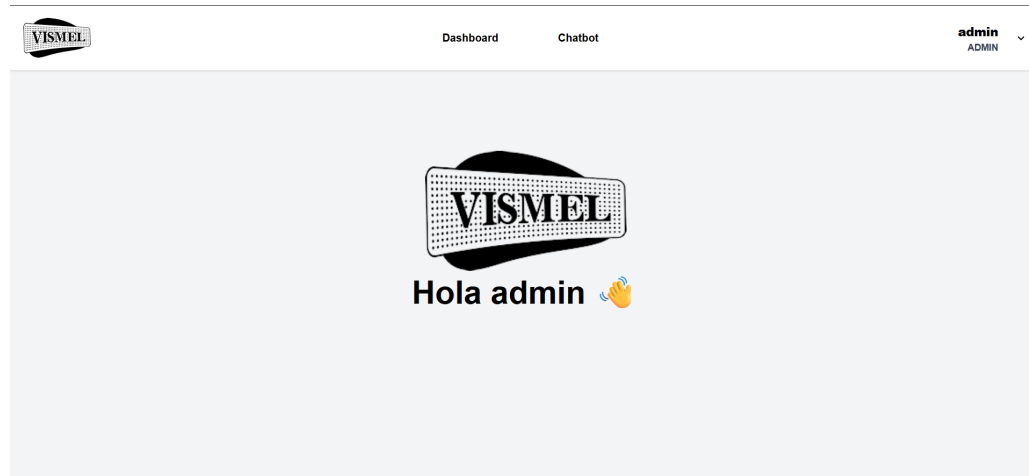


Figura 3.4: Pàgina Home

Si a la barra de navegació es pitja el botó de *dashboard* s'obrirà aquesta mateixa pàgina (3.5, 3.6) amb una breu demostració de les dades de la BD mostrades de distintes formes. El primer gràfic és un circular que mostra el nombre de màquines que hi ha dividides pel seu tipus, el segon és un gràfic de barres horitzontal amb les avaries totals dividides per prioritat i per si estan completades o no. El següent és un mapa de les Illes Balears amb la ubicació dels clients de Vismel. I finalment una llista amb les pròximes revisions a realitzar.

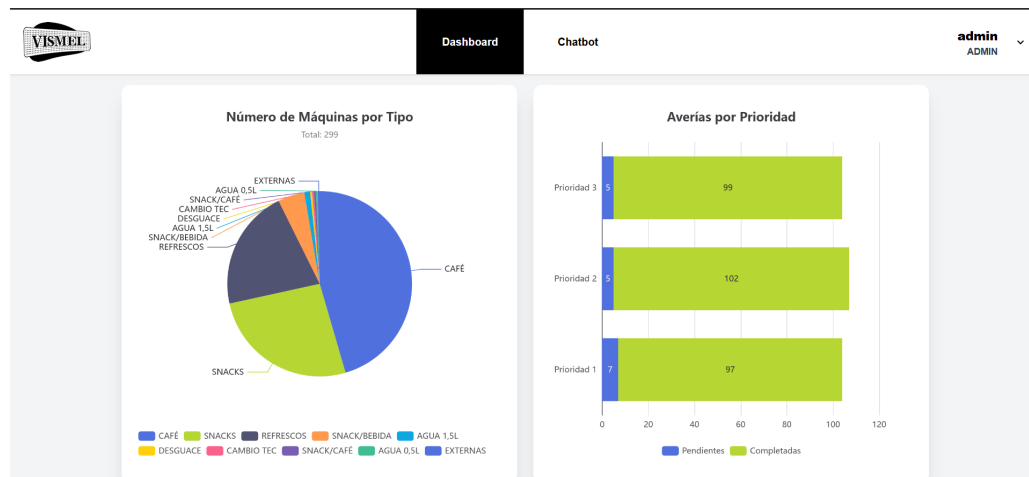


Figura 3.5: Pàgina Dashboard: Elements 1 i 2

### 3. DISSENY I REQUISITS

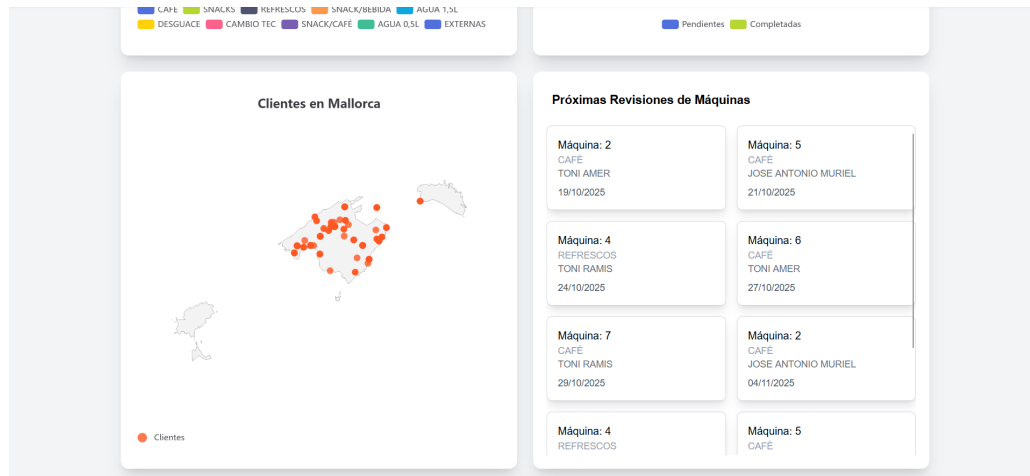


Figura 3.6: Pàgina Dashboard: Elements 3 i 4

En canvi, si a la barra de navegació es pitja el botó de *Chatbot* llavors es mostrarà el xat de l'usuari amb el bot de l'empresa (3.7). A la dreta es mostren els missatges enviats per l'usuari, mentre que a l'esquerra es veuen les respostes del bot (si és que en retorna) les quals poden contenir text en Markdown que és processat per poder mostrar de forma més dinàmica la resposta (per exemple, text en negreta o llistes). A la part inferior es mostra un quadre de text on l'usuari pot escriure la seva consulta. Després té un botó a la dreta per enviar el missatge (o pitjant la tecla *enter*), i per fer neta la *cache* i esborrar l'historial (equivalent a començar un nou xat) tens el botó corresponent a l'esquerra.



Figura 3.7: Pàgina Chatbot

Finalment, en pitjar el desplegable de l'usuari (3.8), es mostra la funció de tancar la sessió la qual fa que retornis a la pàgina d'inici de sessió (3.3) i la funció *Usuarios* que mostra la pàgina.



Figura 3.8: Desplegable amb Funcions de l'Usuari

En aquesta pàgina de *users* (3.9) es mostren tots els usuaris de l'aplicació i des d'on es poden gestionar afegint usuaris, eliminant usuaris, o canviant el seu rol dins l'aplicació. Els usuaris estan disposats en una taula mostrant les seves dades (menys la contrasenya) i per a cada un tens: un desplegable que mostra el rol actual i permet canviar-lo a un rol diferent, i un botó a la dreta per eliminar-lo. Per guardar tots els canvis es pot pitjar el botó corresponent a la part inferior esquerra.

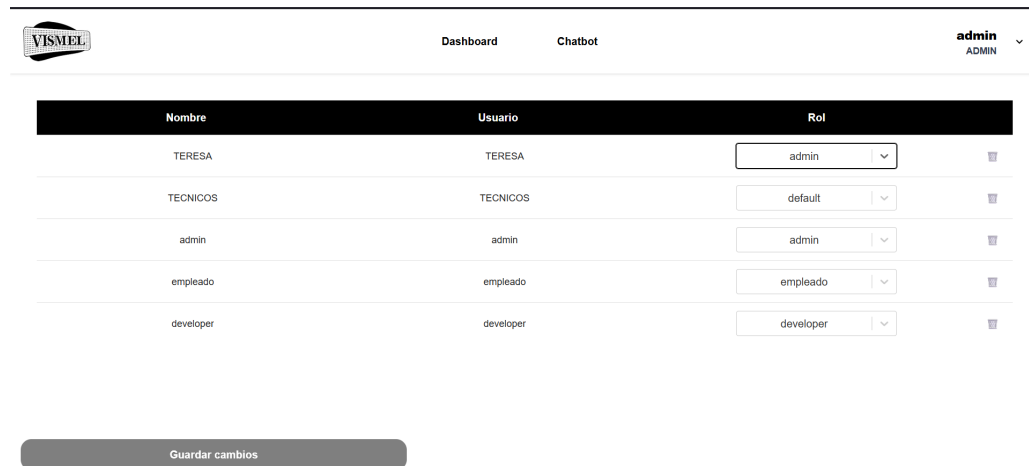


Figura 3.9: Pàgina Users

## 3.5 Comunicació entre Mòduls

D'ençà que es decideix separar l'aplicació en mòduls s'ha de dissenyar la manera en què es comuniquen les diferents tecnologies. S'han d'obviar els mòduls de les BDs perquè aquests es comuniquen a través de les corresponents llibreries segons la tecnologia emprada.

### 3.5.1 Frontend -> Backend Node.js

La primera interacció que sempre es dona en usar l'aplicació és la comunicació entre el *frontend* i el *backend* de Node.js. Aquestes interaccions es poden donar tant per obtenir informació (com el cas per obtenir l'historial del xat d'un usuari) o per enviar informació com clarament es fa quan s'envia una consulta des del xat. La comunicació en aquest cas és molt simple, ja que el *frontend* només fa peticions a les APIs del *backend* les quals tenen un disseny *Representational State Transfer* (Transferència

d'Estat Representacional) (REST), emprant el protocol HTTP i mètodes com *GET*, *POST* o *PATCH*. Les APIs emprades són:

- *Autenticació*
  - **POST /api/auth/login**: Per l'inici de sessió amb un nom d'usuari i una contrasenya.
  - **POST /api/auth/register**: Per registrar un usuari amb un nom, un nom d'usuari, una contrasenya i el seu rol.
- *Xat*
  - **POST /api/chat/talk**: Emprat per enviar un missatge al bot i rebre una resposta.
  - **GET /api/chat/talk**: Per obtenir l'historial de missatges d'un usuari amb el bot.
  - **DELETE /api/chat/talk**: Per eliminar l'historial de missatges i començar un nou xat.
- *Dashboard*
  - **GET /api/dashboard/:id**: Amb aquest *endpoint* es pot obtenir informació emprada en un dels panells de la pàgina *dashboard* identificat per un *id*.
- *Usuaris*
  - **GET /api/users**: Emprat per obtenir la informació completa de tots els usuaris de l'aplicació.
  - **GET /api/users/roles**: S'usa per obtenir el llistat de tots els rols que hi ha.
  - **PATCH /api/users**: Per actualitzar un o diversos usuaris de la BD segons unes condicions que poden anar desde el seu nom d'usuari com el rol que posseeix.
  - **PATCH /api/users/:id**: Per actualitzar un sol usuari només a partir del seu *id*, això per agilitzar l'edició quan només es tracta d'un sol usuari a editar.

#### 3.5.2 Backend Node.js -> Backend Python

Les comunicacions entre els dos *backends* només es donen en el cas que l'usuari envii una consulta des del xat i s'usi l'API abans mencionada. Quan això ocorre el *backend* de Node.js envia el missatge (junt amb altres paràmetres com l'historial) a l'API del *backend* de Python la qual és l'única en tot el mòdul perquè aquest només s'encarrega de la comunicació amb el bot. L'*endpoint* mencionat és **POST /agent/talk** el qual rep el missatge i l'historial del xat i inicia el procés per enviar la consulta a l'API de OpenAI, en acabar retorna la resposta del bot.

### 3.5.3 Flux de Comunicació amb l'LLM d'OpenAI

Quan es rep el missatge de l'usuari des de l'API corresponent s'inicia un flux on s'emprà varies cridades a l'API de OpenAI amb diferents prompts dissenyat per obtenir la millor resposta possible per a l'usuari.

1. El primer que es fa és comprovar la connexió amb la BD. Si és correcte segueix el flux, sinó s'executa la funció `build_answer_without_query` per obtenir una resposta fora comptar amb dades de la BD.
2. Assegurant la connexió, s'executa la funció `get_response` que és la funció general que permet seguir tres camins: generar una resposta directa a la pregunta en cas de no necessitar connexió amb la BD, generar una consulta SQL per obtenir dades o generar una consulta SQL prèvia per obtenir dades que ens ajudin a generar la consulta SQL definitiva.[16]
3. Tant si es necessita la consulta SQL bàsica com la prèvia aquestes s'executaran a la BD, si hi ha resultats segueix el flux, sinó s'executa `build_answer_without_query`.
4. Si hem recuperat les dades de la BD llavors executarem la funció `build_answer_from_query` si es tracta de la consulta normal que donarà una resposta amb el context extra de les dades, o `get_query_from_previous_data` si es tracta de la consulta prèvia.
5. En el cas d'executar `get_query_from_previous_data` amb les dades recuperades de la primera consulta aquest ens pot donar en rares ocasions una resposta directa (si la informació de l'anterior consulta no és suficient) o, amb més normalitat, genera la consulta SQL definitiva per extreure les dades necessàries per respondre al missatge de l'usuari.
6. Finalment la consulta SQL generada per `get_query_from_previous_data` s'executa a la BD, si s'obtenen resultats s'executarà `build_answer_from_query`, sinó s'executarà `build_answer_without_query`.

En cas que hi hagi errors per dins el flux sempre hi haurà un *callback* a la funció `build_answer_without_query` per assegurar una resposta a l'usuari, encara que aquesta no sigui la idònia. Tot això es pot veure millor a la figura 3.10.

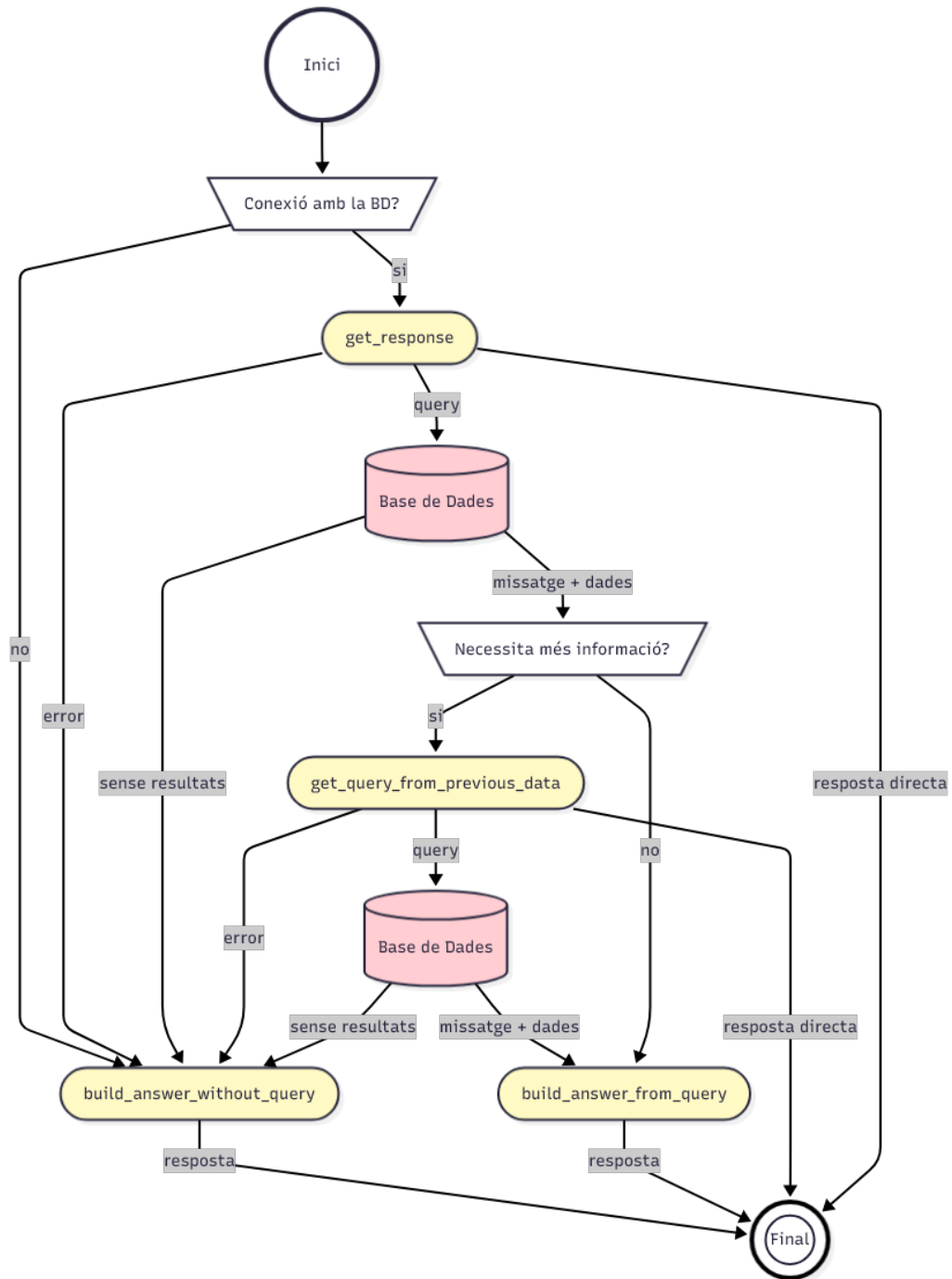


Figura 3.10: Diagrama de Flux del Backend de Python

Els prompts dissenyats que s'adjunten a les consultes de l'API d'OpenAI a cada funció són:

1. **Prompt Global:** aquest és comú a tots els prompt usats en totes les funcions.



Listing 3.1: Prompt General

```

1  """
2  You are a bot working for Vismel, a vending machine company in
   Inca, Mallorca. You act as the company's assistant,
   accessing the database and all its records to improve your
   responses.
3
4  - Current date and time: {current_date}
5  - Always respond in {lang}.
6  - Always use the International System of Units (SI).
7  - Always work in euros. Mark all prices and monetary amounts
   with the € symbol (e.g., 1000 €).
8  - Use a comma (,) as the decimal separator in textual responses
   (e.g., 10,50). In SQL queries, always use a dot (.) for
   decimal numbers (e.g., 0.1).
9  - All output must be valid JSON. Do not include markdown, HTML,
   quotes, or explanatory text outside the JSON structure.
   All textual answers go in the 'response' field.
10 - The 'response' field must contain a **flat string**, not
   nested structures.
11 - Use **basic Markdown formatting** by default, always favoring
   clarity and simplicity:
12   - Use '-' for bullet lists.
13   - Use line breaks ('\n') between list items or paragraphs.
14   - Avoid Markdown tables unless strictly necessary.
15   - Do not include raw HTML tags (e.g., '<br>', '<div>', '<span
   >').
16 - When returning values, include units (€, %, etc.) clearly.
17 - Structure responses to be readable and organized, but avoid
   excessive formatting.
18 - Never return nested JSON, arrays, or dictionaries inside '
   response'. All content must be a flat string.
19 - Use names and descriptions instead of internal IDs or codes
   where possible.
20 - Always assume that previous context is relevant unless the
   user states otherwise.
21 - If input is ambiguous (e.g., partial or fuzzy matches), infer
   intent intelligently. E.g., a 'nombre' column likely
   includes full names.
22 - Keep responses within {limit} characters unless more detail
   is required for clarity or accuracy.
23 - You are an assistant for a vending machine company. You have
   access to its database, provided under 'db_schema', and
   some static tables listed below.
24 - Check if the user's question can be fully or partially
   answered using 'static_tables'. Only consider querying the
   database if no relevant static information is found.
25 - If the static data is sufficient, avoid generating SQL and
   provide the answer directly.
26 - Ensure that all output is valid JSON. Avoid partial or
   malformed JSON. If unsure, do not output anything.
27 - Static tables include: {static_tables_description}
28 """

```

2. **Common Data:** aquest prompt és només per `get_response` i `get_query_from_previous_data`, i afegeix informació de la BD com l'esque-

ma o les taules estàtiques per poder generar millors consultes SQL.

Listing 3.2: Prompt amb Informació de la Base de Dades

```
1  """
2  Always analyze this following data, even if it is contained in
   a file, as it may contain relevant information to improve
   responses.
3  <schema>
4  This is the schema of the database:
5
6  {db_schema}
7  </schema>
8  <static_tables>
9  These are some tables from the database, use it to create
   better queries or avoid the execution of an unnecessary
   query:
10
11  {static_tables}
12  </static_tables>
13
14  <tips>
15  - Use data from static_tables **only** for metadata (roles,
   familias, modelos, tipos, frecuencias).
16  - Never use static_tables as source for runtime values (stock,
   precio, cantidades, recaudaciones).
17  - Only use 'similarity()' if input is ambiguous or fuzzy.
18  - Validate all output JSON before returning it.
19  - Always explain reasoning clearly in 'response'.
20  </tips>
21  """
```

3. **get\_response**: prompt per obtenir una resposta directa a la consulta de l'usuari, una consulta SQL bàsica o una consulta SQL per obtenir informació prèvia i generar una millor consulta i obtenir les dades correctes de la BD.

Listing 3.3: Prompt de la funció get\_response

```
1  """
2  Analyze the user's question and determine if accessing the
   database is necessary to provide an accurate answer, or if
   you can answer directly with static data or simple
   inferences.
3
4  ### Rules:
5
6  1. If the question can be answered using static tables (roles,
   types, families, etc.), answer directly. Do not generate
   SQL.
7  2. If the question requires variable data (stock, prices,
   quantities, collections, etc.), generate an SQL query on
   the pivot tables with a vismel prefix.
8  3. Before generating any condition in the SQL query, always:
9     - Check the **data type** of the column in the database
   schema.
10    - If the column is of type 'text', 'char', 'varchar', 'bit',
   'boolean', or any non-numeric type:
```

```

11 · Always use **case-insensitive comparison** ('ILIKE' or
    'LOWER(field) = LOWER('input')') when checking
    equality.
12 · If the user input is likely free-text, fuzzy, or human-
    entered (names, ciudades, descripciones, etc.), **
    never** use direct equality ('=') alone.
13 · Instead:
14   1. Prefer 'ILIKE '%input%'' when the intent is
    substring or contains search.
15   2. Or use 'LOWER(field) = LOWER('input')' when strict
    equality is acceptable but case-insensitive.
16   3. If ambiguity remains, generate an 'extra_sql_query'
    with 'similarity(field, 'input') > 0.3' ordered
    by descending similarity.
17 · Example:
18   ```
19   SELECT cliente.nombre
20   FROM vismel.cliente
21   INNER JOIN vismel.poblacion ON cliente.poblacion_id =
    poblacion.codigo_postal
22   WHERE LOWER(poblacion.nombre) = LOWER('Inca')
23   LIMIT 3;
24   ```
25   or with substring search:
26   ```
27   WHERE poblacion.nombre ILIKE '%inca%'
28   ```
29 4. If the question requires variable data (stock, prices,
    quantities, collections, etc.), generate an SQL query on
    the pivot tables with the full schema prefix 'vismel.' on
    all tables and columns.
30 5. If the question references any **user-entered, human-written
    , or free-text** fields - such as:
31   - Names (e.g., user names, employee names),
32   - Descriptions or titles,
33   - Comments or observations,
34   - Textual identifiers in fuzzy or informal formats,
35   then:
36   - **Do not generate a direct SQL query** immediately.
37   - First, generate a similarity search using 'similarity(
    field, 'input') > 0.3', ordered by descending similarity
    .
38   - Only after retrieving possible matches should a final
    query be created, based on the most likely match (
    typically similarity > 0.7).
39 6. Never assume the input is perfectly written. Always verify
    fuzzy fields with 'similarity()' before declaring no
    results.
40 7. If valid candidates are identified in the similarity search,
    generate the main query using the most appropriate match.
41 8. If the question involves a prediction or estimate (e.g., "
    How much will be collected?"), use historical data to
    calculate:
42   - The average
43   - The year-over-year trend or growth
44   - A reasoned and justified projection
45 9. If there is not enough data to answer the question (for

```

### 3. DISSENY I REQUISITS

---

```
example, a lack of records), indicate this clearly without
inventing anything.
46 10. If the field used in filters, sorts, or groupings can
    contain NULL values (e.g., 'coste', 'precio', 'cantidad'),
    and the query logic involves comparing or ranking those
    values:
47   - Exclude records with NULL values using a 'WHERE IS NOT NULL
      ' field.
48   - This is especially important for rankings, sums, averages,
      and filters by maximum or minimum.
49   - Don't display records with NULL values if it breaks the
      logic or displays meaningless results.
50 11. Never nest aggregate functions directly (e.g., avoid 'MIN(
    ABS(SUM(...)))' or 'AVG(SUM(...))'). Instead:
51   - First, write a subquery or CTE that computes the inner
      aggregation, for example:
52   '
53   SELECT maquina_id, SUM(importe_recaudado) AS total_recaudado
54   FROM vismel.recaudacion_semanal
55   GROUP BY maquina_id
56   '
57   - Then, in an outer query, apply the aggregate function:
58   '
59   SELECT maquina_id, total_recaudado
60   FROM (
61     -- inner aggregation
62   ) AS sub
63   WHERE ABS(total_recaudado - 1000) = (
64     SELECT MIN(ABS(total_recaudado - 1000)) FROM (
65       -- inner aggregation
66     ) AS sub2
67   )
68   '
69   - Using this pattern avoids nested aggregates and the
      related SQL errors.
70
71 ### Expected Output:
72
73 Returns a valid JSON with these keys:
74
75 - "sql_query": Main query that answers the question, if
    possible.
76 - "response": Direct textual response if no SQL is required or
    if it can be reasonably inferred.
77 - "extra_sql_query": Similarity query, if necessary, to resolve
    ambiguity.
78
79 ### Examples:
80
81 Example 1 - Direct SQL:
82 {{
83 "sql_query": "SELECT * FROM vismel.usuario WHERE nombre = 'Juan
    '";
84 "response": "",
85 "extra_sql_query": ""
86 }}
87
```

```

88 Example 2 - Similarity Search:
89 {{
90 "sql_query": "",
91 "response": "",
92 "extra_sql_query": "SELECT nombre, similarity(nombre, 'toni
    camp') AS similarity FROM vismel.usuario WHERE similarity(
    nombre, 'toni camp') > 0.3 ORDER BY similarity DESC;"
93 }}
94
95 Example 3 - Prediction:
96 {{
97 "sql_query": "SELECT year, week, SUM(amount) AS total FROM
    vismel.weekly_collection WHERE week = 30 GROUP BY year,
    week ORDER BY year DESC;",
98 "response": "",
99 "extra_sql_query": ""
100 }}
101
102 Example 4 - Using static data:
103 {{
104 "sql_query": "",
105 "response": "El rol del usuario llamado 'Juan' es 'técnico'.",
106 "extra_sql_query": ""
107 }}
108 ""

```

4. `get_query_from_previous_data`: prompt per obtenir una consulta SQL millorada a partir d'unes dades d'una consulta prèvia, si no és possible generar la nova consulta genera una resposta directa.

Listing 3.4: Prompt de la funció `get_query_from_previous_data`

```

1  ""
2  The user's question involved ambiguous input (e.g., a partial
    or imprecise name). A prior similarity-based query was
    executed to obtain potential matches.
3
4  Your task is now to analyze:
5
6  1. The user's original question.
7  2. The similarity query and its result.
8  3. The database schema.
9
10 Determine whether:
11 - You can identify with sufficient confidence (e.g., top result
    with similarity > 0.3) the correct record the user is
    referring to.
12 - If yes, generate a definitive SQL query using the 'vismel.'
    schema prefix and place it in the 'sql_query' field.
13 - If not, respond directly based on the previous data or
    clarify the ambiguity in the 'response' field.
14 - If no reasonable answer or query can be produced, leave all
    fields empty.
15 - If multiple tables are needed, use joins based on foreign key
    relations or inferred logic.
16 - If the user mentions a person (employee, admin, etc.), assume
    it refers to a record in 'vismel.usuario'.

```

### 3. DISSENY I REQUISITS

---

```
17 - Consider if the top result from the similarity search is
    confidently correct (e.g., similarity > 0.7 and clearly
    matches intent).
18 - If multiple candidates are possible, do not choose
    automatically. Instead, ask the user to clarify or show
    potential matches.
19 - Do not guess a definitive query unless the match is strong
    and unambiguous.
20 - Always review the data types from the schema before building
    SQL conditions. Use quotes for text/char/bit fields.
21 - If valid candidates are identified in the similarity search,
    generate the main query using the most appropriate match.
22 - If the question involves a prediction or estimate (e.g., "How
    much will be collected?"), use historical data to
    calculate:
23     - The average
24     - The year-over-year trend or growth
25     - A reasoned and justified projection
26 - If there is not enough data to answer the question (for
    example, a lack of records), indicate this clearly without
    inventing anything.
27 - If the field used in filters, sorts, or groupings can contain
    NULL values (e.g., 'coste', 'precio', 'cantidad'), and the
    query logic involves comparing or ranking those values:
28     - Exclude records with NULL values using a 'WHERE IS NOT NULL
      ' field.
29     - This is especially important for rankings, sums, averages,
      and filters by maximum or minimum.
30     - Don't display records with NULL values if it breaks the
      logic or displays meaningless results.
31 - Never nest aggregate functions directly (e.g., do not use '
    AVG(SUM(...))'. Instead:
32     - First, use a subquery to compute the inner aggregation (e.
      g., 'SUM(...) GROUP BY ...').
33     - Then apply the outer aggregation (e.g., 'AVG(...)') on
      that subquery.
34     - This avoids SQL errors like 'aggregate function calls
      cannot be nested'.
35
36
37 Output valid JSON with:
38 - "sql_query": Final SQL query if identifiable.
39 - "response": Textual answer if no SQL can be created.
40
41 ### Examples:
42
43 Example 1 - Final query identified:
44 {{
45 "sql_query": "SELECT * FROM vismel.usuario WHERE nombre = 'Juan
      '";",
46 "response": ""
47 }}
48
49 Example 2 - No query generated:
50 {{
51 "sql_query": "",
52 "response": "The largest ocean is the Pacific Ocean."
```

```

53 }}
54
55 Example 3 - Ambiguous name, ask user:
56 {{
57   "sql_query": "",
58   "response": "Se encontraron múltiples coincidencias para '
               toni camp'. ¿Podrías especificar a cuál te refieres?"
59 }}
60
61
62 <previous_sql_query>
63 This is the intermediate query that has been requested to be
   executed to obtain results and improve the context:
64
65 {previous_sql}
66 </previous_sql_query>
67 <previous_sql_result>
68 These are the results obtained from the intermediate query and
   which improve the context:
69
70 {previous_result}
71 </previous_sql_result>
72 ""

```

5. **build\_answer\_from\_query**: prompt per obtenir una resposta al missatge de l'usuari a partir de les dades obtingudes d'una consulta SQL prèvia.

Listing 3.5: Prompt de la funció `build_answer_from_query`

```

1  ""
2  Based on the user's question and the result of a previously
   executed SQL query, provide a clear, concise, and accurate
   response. Use the query structure and database schema to
   ensure the answer is well-informed.
3  Also verify that the result set does not include NULL values in
   key fields (like cost, price, stock, or revenue) unless
   they are specifically relevant. If the logic of the user's
   question implies comparison or ranking, NULLs should be
   excluded from the SQL query.
4
5  - When using the result of a SQL query, interpret all relevant
   columns and summarize the answer in clear and human-
   friendly form.
6  - Do not just restate the raw result. Explain it naturally.
7  - If the result is empty or lacks relevance, clarify it.
8  - When interpreting query results or confirming query logic,
   always check the column types in the schema. Ensure that
   conditions match their proper type (e.g., text fields are
   quoted).
9  - Format the 'response' using basic Markdown if it improves
   readability. Favor '-' bullet points and natural line
   breaks. Avoid HTML or complex structures.
10
11 <sql_query>
12 This is the query that was executed to obtain the results:
13
14 {sql_query}

```

### 3. DISSENY I REQUISITS

---

```
15 </sql_query>
16 <sql_response>
17 These are the results obtained from the query that was
    requested to be executed based on the user's question:
18
19 {result}
20 </sql_response>
21 ""
```

6. `build_answer_without_query`: prompt per obtenir una resposta al missatge de l'usuari fora informació addicional que potser és necessària.

Listing 3.6: Prompt de la funció `build_answer_without_query`

```
1 """
2 Given a user's question, provide a direct, clear, and
    informative response. Do not fabricate data. It's possible
    a query was attempted before but didn't return usable
    results, or the question may not require any data retrieval
    .
3 You are provided with the previous SQL query and schema in case
    referencing them could help improve or refine your answer.
4 Use the schema or failed query only if they clearly help to
    explain *why* no data was found or what might be wrong (e.g
    ., a typo or missing record).
5 Do not speculate or fabricate information. If unsure, clearly
    indicate the limitation.
6
7 - Format the 'response' using basic Markdown if it improves
    readability. Favor '-' bullet points and natural line
    breaks. Avoid HTML or complex structures.
8
9 <sql_query>
10 This is the possible query that was attempted to be executed
    but from which no results were obtained:
11
12 {sql_query}
13 </sql_query>
14 """
```

## 3.6 Estructura de Carpetes del Projecte

Per aconseguir un disseny escalable, cal una bona organització de carpetes, separant clarament cada mòdul i les funcions que conté. A l'arrel del projecte (3.11) es poden trobar fitxers compartits o generals com el `docker-compose.yml` que permet construir i alçar tots el contenidor alhora. També trobem un executable (`docker-clean-build`) que agilitza l'inici complet i net del projecte. Però el més important és el fitxer `.env.template` que serveix com a base per configurar les variables globals necessàries del projecte com les adreces *Internet Protocol* (Protocol d'Internet) (IP), ports utilitzats pels contenidors, contrasenyes, etc.

Les carpetes principals del projecte són: `backend`, `frontend` i `db` (BD). Dins la carpeta *backend* hi ha les subcarpetes corresponents a `node` i `python`.



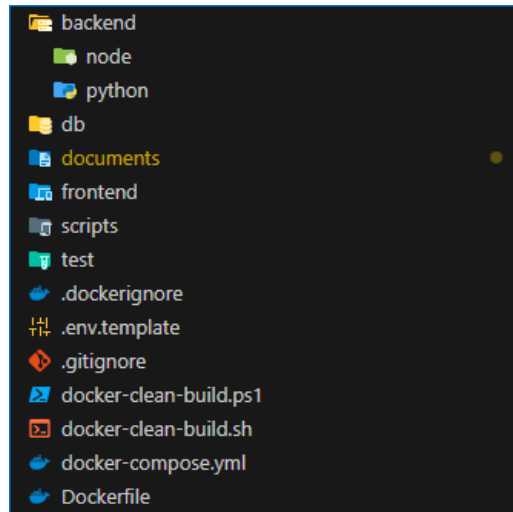


Figura 3.11: Estructura d'Arxius: Arrel

#### 3.6.1 Carpeta Frontend

Aquí trobem tots els fitxers relacionats directament amb la interfície gràfica, com mostra la figura 3.12. A l'arrel (veurem que a tots els mòduls l'estructura és similar) hi ha arxius bàsics de la configuració de Vite+React entre els quals destaquen: el `package.json` amb informació del projecte i les seves dependències, i `vite.config.js` que és l'arxiu principal amb les configuracions i variables d'aquest mòdul.

Dins la carpeta `src` hi ha el codi font de mòdul separat per carpetes segons les funcionalitats o característiques:

- `api`: Codi encarregat de les consultes a les APIs del *backend*.
- `assets`: Arxius estàtics que no son codi com imatges.
- `components`: Components reutilitzables de la interfície.
- `constants`: Arxius amb valors fixos compartits dins el mòdul.
- `context`: Contexts personalitzats per l'estat global, com per exemple, l'estat d'autenticació.
- `hook`: Lògica reutilitzable, similar als components però no visuals.
- `layout`: Defineixen l'estructura general o de parts grans de l'aplicació.
- `pages`: Pàgina completa dins l'aplicació, com per exemple, la pàgina *home*.
- `routes`: Rutes per a la navegació del projecte.
- `utils`: Funcions auxiliars.

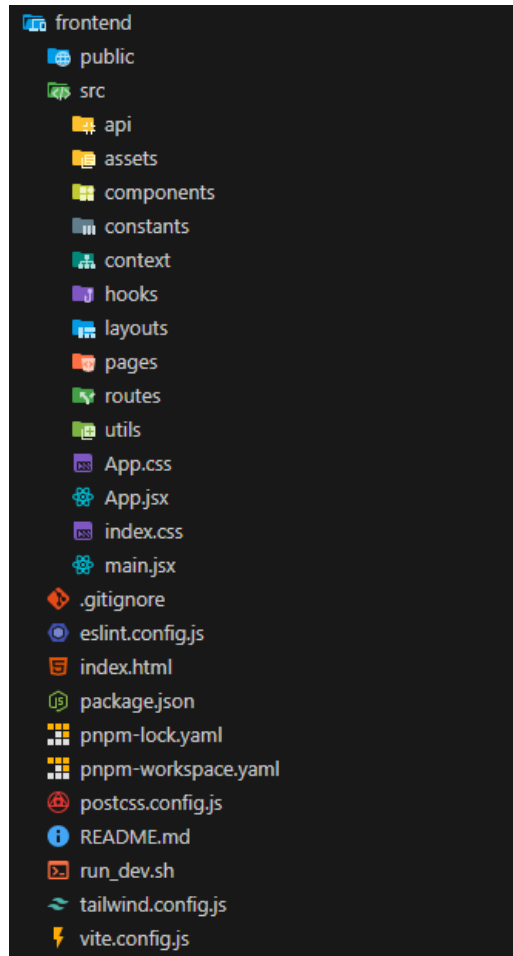


Figura 3.12: Estructura d'Arxius: Frontend

#### 3.6.2 Carpeta Backend Node

L'arrel d'aquesta carpeta (3.13) conte els arxius de configuració com el `package.json` i el `Dockerfile` per a la configuració del contenidor entre altres.

La carpeta `src` té la següent estructura del codi font:

- `api`: Defineix les rutes i *endpoints* de les funcionalitats exposades per l'API.
- `config`: Arxius de configuració compartida dins el mòdul.
- `constants`: Arxius amb valors fixos compartits dins el mòdul.
- `controllers`: Arxius que es vinculen a les rutes definides i s'encarreguen de gestionar la lògica de cada petició. Reben les dades, les processen i retornen una resposta.
- `db`: Codi relacionat amb la connexió i gestió de les bases de dades.

- `logging`: Implementació d'un registre de *logs* personalitzat per desenvolupament i manteniment.
- `middleware`: Funcions intermèdies executades entre la petició i la resposta.
- `services`: Lògica de negoci principal.
- `utils`: Funcions auxiliars.
- `index.js`: Fitxer principal del projecte on es configura i inicia el servidor.

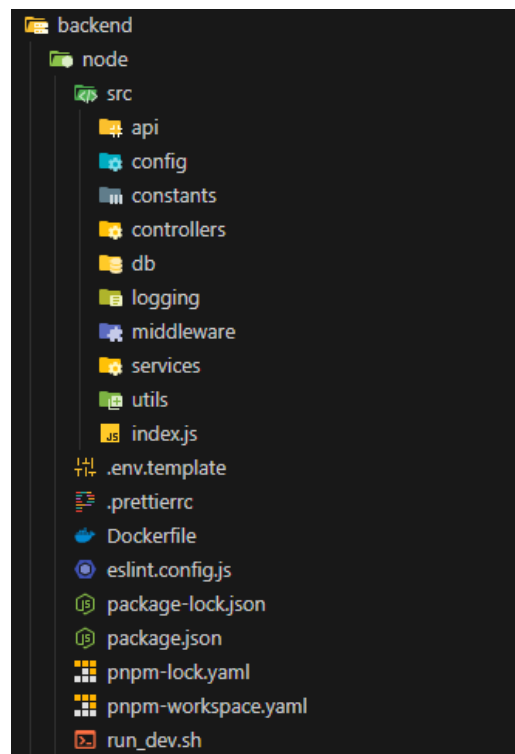


Figura 3.13: Estructura d'Arxius: Backend Node

#### 3.6.3 Carpeta Backend Python

Aquí trobem el codi font per a la comunicació amb l'API d'OpenAI. A l'arrel (3.14) destacable només trobem el `Dockerfile` per la configuració del contenidor i el fitxer `requirements.txt` amb les dependències del mòdul.

A la carpeta `src` trobem:

- `config`: Arxius de configuració compartida dins el mòdul.
- `constants`: Arxius amb valors fixos compartits dins el mòdul.
- `db`: Codi relacionat amb la connexió i gestió de les bases de dades.

- **logging:** Implementació d'un registre de *logs* personalitzat per desenvolupament i manteniment.
- **models:**
- **routes:** En aquest cas només defineix l'*endpoint* de l'API d'aquest mòdul per accedir a la comunicació amb el bot.
- **services:** Lògica de negoci principal. Aquí es troben els fitxers que s'encarreguen de comunicar-se amb l'API d'OpenAI.
- **utils:** Funcions auxiliars.
- **main.py:** Fitxer principal del projecte on es configura i inicia el servidor.

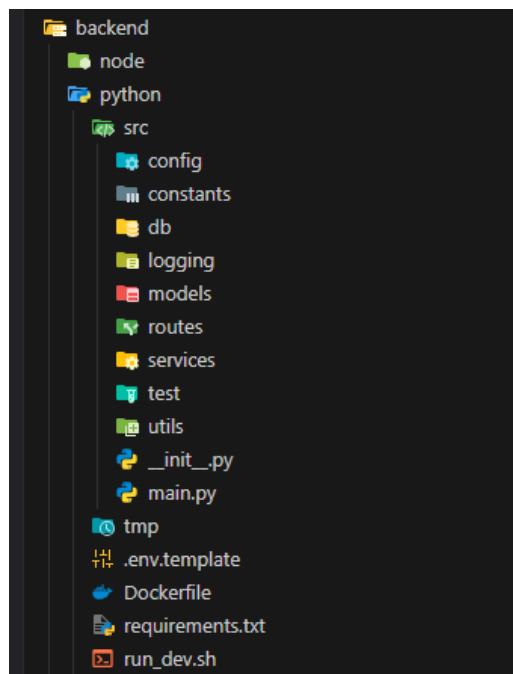


Figura 3.14: Estructura d'Arxius: Backend Python

#### 3.6.4 Carpeta de Bases de Dades

Finalment, a la carpeta `db` (3.15) trobem arxius relacionats tant amb la BD de PostgreSQL com amb la de Redis.

- **data:** Arxius que contenen informació sobre la BD principal.
  - **data\_insertion:** Conté totes les dades usades durant el desenvolupament del projecte. **Aquestes dades han estat inventades i no corresponen amb les dades reals de l'empresa.**

- extra: Arxius amb el codi SQL necessari per crear estructures i funcionalitats addicionals a la BD com descripcions de taules i columnes, o creació de vistes.
- table\_creation: Codi SQL necessari per crear les taules de la BD i les relacions entre elles.
- postgres-init: Codi SQL executat a l'inici de la creació del contenidor de PostgreSQL i que inicialitza la configuració de la BD i insereix la informació necessària.
- redis: Arxius de configuració del contenidor i de la BD de Redis.

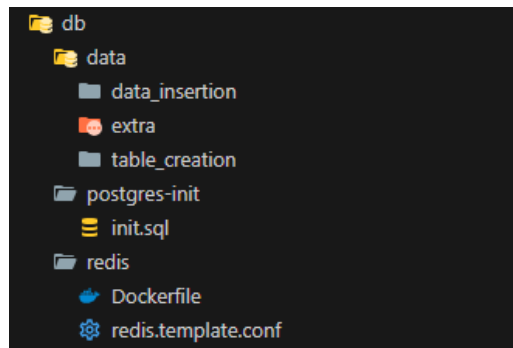


Figura 3.15: Estructura d'Arxius: Bases de Dades



## IMPLEMENTACIÓ

### 4.1 Introducció a la Implementació

En aquest capítol es descriu el procés de materialització del disseny presentat prèviament, detallant com s'han desenvolupat i integrat els diferents mòduls que conformen el sistema. L'objectiu és mostrar de manera pràctica les decisions preses durant el desenvolupament, la configuració de l'entorn i el codi més rellevant. A més, es presenten les dificultats que han anat sorgint i com s'han resolt.

### 4.2 Entorn de Desenvolupament i Producció

Com s'ha anat mencionant, amb Docker s'han construït els contenidors de cada part de l'aplicació. Això s'aconsegueix gràcies als `Dockerfiles` de cada mòdul i un `docker-compose.yml` que unifica tota la configuració i permet un desplegament més àgil i centralitzat, a més facilita la comunicació entre contenidors gràcies a la seva xarxa interna.

La primera capa és un `.env` que defineix la configuració global de tota l'aplicació. Aquest ha de ser únic i personalitzat en cada instància diferent del projecte. Pots trobar:

1. Adreça IP (normalment sempre s'utilitza `localhost` o un domini si és necessari `HTTPS`) i els ports que usarà cada contenidor.
2. Rutes de les APIs. Es pot personalitzar una part d'aquestes però no és recomanable.
3. Configuració de les BDs amb la adreça IP, el port i credencials entre altres coses.
4. Clau interna per la comunicació entre els dos *Backend*. Això permet una capa més de seguretat.

## 4. IMPLEMENTACIÓ

5. Model de OpenAI que s'usa i el seu límit de tokens per indicar quan s'ha d'usar l'adjunció de fitxers.
6. Entorn actual (desenvolupament o producció). El més rellevant d'aquesta variable és que indica com s'imprimiran els *logs*.

Per a producció el més recomanat és usar els contenidors de Docker, encara que es podria tenir una màquina executant els processos dels servidors. Per als contenidors el `docker-compose.yml` es divideix en els 5 serveis de l'aplicació que es configuren segons les variables indicades al `.env`:

- **postgres:** Empra la darrera imatge estable de PostgreSQL i en iniciar empra l'arxiu SQL de `db/postgres-init` per inicialitzar la BD amb l'esquema de taules corresponent i les dades a inserir (en cas de producció real les dades s'han d'eliminar o emprar les correctes amb les quals es vol posar en marxa el projecte). No és necessari `Dockerfile`.

Listing 4.1: Servei postgres dins l'arxiu `docker-compose.yml`

```
1 postgres:
2   image: postgres:latest
3   container_name: postgres
4   restart: always
5   environment:
6     POSTGRES_USER: ${DB_USER}
7     POSTGRES_PASSWORD: ${DB_PASSWORD}
8     POSTGRES_DB: ${DB_NAME}
9   ports:
10    - "${DB_PORT}:${DB_PORT}"
11   volumes:
12    - "pg_data:/var/lib/postgresql/data"
13    - "./db/postgres-init:/docker-entrypoint-initdb.d"
```

- **redis:** Per configurar aquest servei s'empra el `Dockerfile` dins `db/redis` i empra la darrera versió estable de Redis 7.

Listing 4.2: Servei redis dins l'arxiu `docker-compose.yml`

```
1 redis:
2   build:
3     context: .
4     dockerfile: ./db/redis/Dockerfile
5   container_name: redis
6   restart: always
7   ports:
8     - "${REDIS_PORT}:${REDIS_PORT}"
9   env_file:
10    - .env
11   volumes:
12    - ./db/redis/redis.template.conf:/etc/redis.template.conf
13   entrypoint: ["/bin/sh", "-c"]
14   command: >
15     "envsubst < /etc/redis.template.conf > /etc/redis.conf &&
16     redis-server /etc/redis.conf"
```



Listing 4.3: Dockerfile per el mòdul de Redis

```

1 # Dockerfile.redis
2 FROM redis:7-alpine
3
4 RUN apk add --no-cache gettext

```

- **backend\_python:** Es posa com a context la carpeta backend/python i s'empra el Dockerfile corresponent.

Listing 4.4: Servei backend\_python dins l'arxiu docker-compose.yml

```

1 backend_python:
2   build:
3     context: ./backend/python
4     container_name: backend_python
5     restart: always
6     ports:
7       - "${VITE_BACKEND_PYTHON_PORT}:${VITE_BACKEND_PYTHON_PORT}"
8     environment:
9       - DB_HOST=postgres
10      - REDIS_HOST=redis
11      - PORT=${VITE_BACKEND_PYTHON_PORT}
12     env_file:
13       - ./backend/python/.env
14       - ../.env

```

Listing 4.5: Dockerfile per el mòdul del Backend de Python

```

1 # Usa Python 3.12.3 como imagen base
2 FROM python:3.12.3-alpine
3
4 # Establece el directorio de trabajo dentro del contenedor
5 WORKDIR /src
6
7 # Copia e instala las dependencias de Python
8 COPY requirements.txt .
9 RUN pip install --no-cache-dir -r requirements.txt
10
11 # Copia el código fuente
12 COPY src/ ./src/
13
14 # Establece variables de entorno
15 ENV PYTHONPATH=src \
16     PYTHONDONTWRITEBYTECODE=1 \
17     PYTHONUNBUFFERED=1
18
19 # Expone el puerto donde corre Uvicorn
20 EXPOSE ${PORT}
21
22 # Comando de inicio
23 CMD ["sh", "-c", "uvicorn src.main:app --host 0.0.0.0 --port ${PORT}"]

```

- **backend\_node:** Es posa com a context la carpeta backend/node i s'empra el Dockerfile corresponent.

## 4. IMPLEMENTACIÓ

Listing 4.6: Servei backend\_node dins l'arxiu docker-compose.yml

```
1 backend_node:
2   build:
3     context: ./backend/node
4     container_name: backend_node
5     restart: always
6     ports:
7       - "${VITE_BACKEND_NODE_PORT}:${VITE_BACKEND_NODE_PORT}"
8     environment:
9       - DB_HOST=postgres
10      - REDIS_HOST=redis
11      - FRONTEND_HOST=frontend
12      - BOT_HOST=backend_python
13      - PORT=${VITE_BACKEND_NODE_PORT}
14     env_file:
15       - ./backend/node/.env
16       - ../.env
```

Listing 4.7: Dockerfile per el mòdul del Backend de Node

```
1 # Usa la imagen oficial de Node.js con versión 22.15.0 Alpine
2 FROM node:22.15.0-alpine
3
4 # Establece el directorio de trabajo dentro del contenedor
5 WORKDIR /src
6
7 # Instala pnpm globalmente
8 RUN npm install -g pnpm
9
10 # Copia package.json y pnpm-lock.yaml para instalar
    dependencias
11 COPY package.json pnpm-lock.yaml* ./
12
13 # Instala las dependencias con pnpm
14 RUN pnpm install
15
16 # Copia el código fuente
17 COPY src/ ./src/
18
19 # Expone el puerto en el que la app correrá
20 EXPOSE ${PORT}
21
22 # Comando para iniciar la app
23 CMD ["pnpm", "start"]
```

- **frontend:** Es posa com a context la carpeta frontend i s'empra el Dockerfile de l'arrel del projecte.

Listing 4.8: Servei frontend dins l'arxiu docker-compose.yml

```
1 frontend:
2   build:
3     context: .
4     dockerfile: Dockerfile
5     container_name: frontend
6     restart: always
7     ports:
```

```

8   - "${VITE_FRONTEND_PORT}:${VITE_FRONTEND_PORT}"
9   env_file:
10  - .env
11  environment:
12  - VITE_BACKEND_NODE_HOST=backend_node
13  - PORT=${VITE_FRONTEND_PORT}

```

Listing 4.9: Dockerfile per el mòdul del Frontend

```

1  # --- Etapa build ---
2  FROM node:22.15.0-alpine AS builder
3
4  WORKDIR /src/frontend
5
6  RUN npm install -g pnpm
7
8  # Copiar sólo lo necesario para instalar deps
9  COPY frontend/package.json frontend/pnpm-lock.yaml ./
10
11 # Copiar todo el frontend primero
12 COPY frontend ./
13 COPY .env .env
14
15 # Instalar deps y generar node_modules
16 RUN pnpm install
17
18 RUN pnpm list vite
19 # Construir la app con vite.config.js en frontend/
20 RUN pnpm run build
21
22 # --- Etapa producción ---
23 FROM node:22.15.0-alpine AS runner
24
25 WORKDIR /src/frontend
26
27 RUN npm install -g serve
28
29 # Copiar sólo la carpeta dist generada en la build
30 COPY --from=builder /src/frontend/dist ./dist
31 COPY .env .env
32
33 RUN apk add --no-cache tini
34
35 ENV PORT=${FRONTEND_PORT}
36
37 EXPOSE ${FRONTEND_PORT}
38
39 ENTRYPOINT ["/sbin/tini", "--"]
40 CMD ["sh", "-c", "serve -s dist -l ${PORT:-5173}"]

```

Com s'ha mencionat abans també es pot executar l'aplicació simplement executant els servidors en segon pla o directament en una consola (això és el que fan els executables `run_dev.sh`). Aquest és el mètode que s'ha usat durant el desenvolupament, ja que és la millor manera de veure els *logs* en directe i visualitzar els canvis al moment. També pots tenir les BDs instal·lades directament dins el teu entorn, però això no s'ha previst

en cap moment dins el projecte i la manera recomanada és sempre amb els contenidors de Docker.

### 4.3 Implementació dels Mòduls

Cada mòdul té la seva implementació pròpia, ja que es tracten de tecnologies diferents, però sempre s'intenta mantenir la mateixa estructura i bones pràctiques dins el codi per aconseguir una implementació el més escalable i mantenible possible. Per això es podrà notar que hi ha similituds entre els diferents mòduls.<sup>1,2</sup>

#### 4.3.1 Frontend (Vite + React)

En aquest mòdul veurem com el codi es pot representar com un arbre jeràrquic creant cues de components on les fulles solen ser els components visuals i l'arrel es l'arxiu `App.jsx` que consta de dues parts: `AppRouter` que defineix les rutes (o pàgines) que formen part de la Web App i `AuthProvider` el qual envolta tots els components i és l'encarregat de gestionar el context global de l'autenticació.

Listing 4.10: Codi Frontend: Codi Inicial

```
1 // App.jsx
2 function App() {
3   return (
4     <AuthProvider> // Se envuelve la app con el contexto global de
      autenticación
5     <Toaster position="top-center" />
6     <AppRouter /> // Rutas definidas
7   </AuthProvider>
8 );
9 }
```

A l'arxiu `AuthContext.jsx` s'exposen com a elements globals dinàmics a la resta de components la informació de l'usuari registrat, el seu token, i funcions per iniciar o tancar la sessió. També s'inicia un procés recurrent que revisa que el token no estigui expirat.

Listing 4.11: Codi Frontend: Contexte d'Autenticació

```
1 // context/AuthContext.jsx
2 export const AuthContext = createContext();
3
4 const controlUserToken = (token) => {
5   try {
6     const decoded = jwtDecode(token); // Se decodifica el token
7     if (decoded.exp * 1000 < Date.now()) { // Se establece tiempo
      de expiración
8       logout();
9     }
10  }
11 }
```

<sup>1</sup>La major part del codi mostrat en aquesta secció no correspon a l'arxiu complet, només les parts més essencials o interessants.

<sup>2</sup>En aquesta secció no es menciona directament la implementació de les BDs perquè el seu disseny s'explica a 3.3.1. També durant les seccions d'implementació dels diferents mòduls es mostra com es realitzen les comunicacions i consultes a aquestes.

```

9      return false;
10    } else {
11      setUser({ // Rellenamos la info del usuario con los datos
12                // del token recibido
13                id: decoded.id,
14                username: decoded.username,
15                rol: decoded.rol,
16                name: decoded.name,
17                sessionID: decoded.sessionID,
18              });
19      setTokenSync(token);
20      return true;
21    }
22  } catch (error) {
23    console.error("Token inválido:", error);
24    logout();
25    return false;
26  }
27 };
28
29 const login = async (username, password) => {
30   // Llamada a la API encargada del inicio de sesión con los
31   // datos proporcionados
32   const result = await loginAPI(username, password);
33
34   if (result.success && result.token) {
35     const valid = controlUserToken(result.token);
36     return { success: valid, message: result.message };
37   }
38
39   return result;
40 };
41
42 const logout = useCallback(() => {
43   setUser(null); // Se eliminan los datos del usuario
44   removeTokenSync(); // Se elimina el token
45 }, []);

```

A `AppRouter.jsx` s'han definit les rutes de l'aplicació, cada una fent referència al codi que s'ha de renderitzar. Com veiem hi ha algunes pàgines que serveixen com a suport per millorar l'experiència d'usuari com la de *home* o la *unauthorized*. N'hi ha algunes que estan envoltades pel component `AuthGuard` el qual protegeix amb autorització algunes pàgines com la del xatbot que són exclusives per certs rols.

Listing 4.12: Codi Frontend: Rutes Definides

```

1 // routes/AppRouter.jsx
2 <Route path="/" element={<Login />} />
3 <Route path="/login" element={<Login />} />
4 <Route path="/logout" element={<Logout />} />
5 <Route path="/unauthorized" element={<Unauthorized />} />
6 <Route path="/home" element={<Home />} />
7 <Route
8   path="/dashboard"
9   element={
10     <AuthGuard allowedRoles={["admin", "developer"]}>
11       <Dashboard />

```

## 4. IMPLEMENTACIÓ

```
12     </AuthGuard>
13   }
14 </>
15 <Route
16   path="/users"
17   element={
18     <AuthGuard allowedRoles={["admin", "developer"]}>
19       <Users />
20     </AuthGuard>
21   }
22 </>
23 <Route
24   path="/chatbot"
25   element={
26     <AuthGuard allowedRoles={["admin", "developer"]}>
27       <Chatbot />
28     </AuthGuard>
29   }
30 </>
```

`AuthGuard.jsx` és simplement un codi que s'executa abans de renderitzar les pàgines indicades i s'assegura que l'usuari tingui l'autorització requerida per aquella pàgina, si no és així retornarà la pàgina *home*.

Listing 4.13: Codi Frontend: Protecció amb Autorització

```
1 // components/auth/AuthGuard.jsx
2 useEffect(() => {
3   // Se comprueba si el usuario tiene su rol en la lista de roles
4   // autorizados
5   if (user && allowedRoles.length && !allowedRoles.includes(user.rol
6     )) {
7     toast.error("Acceso Denegado", { id: "acceso-denegado" });
8     setDenied(true);
9   }
10 }, [user, allowedRoles]);
11
12 // Si no está autorizado se redirige a la página Home
13 if (!user) return <Navigate to="/" />;
14 if (denied) return <Navigate to="/home" />;
15
16 return children;
```

També hi ha un `MainLayout.jsx` el qual defineix l'estructura general de la majoria de pàgines, assegurant una barra de navegació i unes mides fixes.

Listing 4.14: Codi Frontend: Estructura General

```
1 // layouts/MainLayout.jsx
2 const MainLayout = () => {
3   return (
4     <div className="min-h-screen flex flex-col">
5       <Navbar /> { /* Barra de navegación */}
6       <div className="h-[6rem]" />
7       <main className="flex flex-col flex-1">
8         <Outlet /> { /* Resto de la página a cargar */}
9       </main>
10     </div>
11   );
12 }
```

```

9     </main>
10   </div>
11 );
12 };

```

La barra de navegació (`Navbar.jsx`) simplement utilitza varis components per incloure els botons de navegació a les pàgines principals i donar una millor UI/UX a la web. El més destacable és la capacitat per mostrar, o no, la navegació a les pàgines depenent de l'autorització de l'usuari, això s'aconsegueix definint els ítems de navegació (dins `navConfig.js`) amb la ruta a la pàgina i els usuaris autoritzats.

Listing 4.15: Codi Frontend: Configuració de la Navegació

```

1 // utils/navConfig.js
2 const navItems = [
3   {
4     to: "/dashboard",
5     label: "Dashboard",
6     roles: ["admin", "developer"],
7   },
8   {
9     to: "/chatbot",
10    label: "Chatbot",
11    roles: ["admin", "developer"],
12  },
13 ];

```

A molt d'arxius es pot veure un ús recurrent del *hook* `useAuth.jsx` el qual simplement s'usa per exportar les variables i funcions globals de `AuthContext.jsx`. Aquest *hook* s'empra en pàgines com la de *Login* per la qual han de passar tots els usuaris per autenticar-se. L'arxiu d'aquesta pàgina simplement renderitza el component `LoginForm.jsx` que a la vegada defineix la funcionalitat d'inici de sessió de `AuthForm.jsx` que crea un formulari amb components com `Input.jsx` o `Button.jsx`.

Listing 4.16: Codi Frontend: Formulari d'Inici de Sessió

```

1 // pages/Login.jsx
2 const Login = () => {
3   const { user, loading } = useAuth(); // Obtener estado de
4     autenticación
5
6   // Mientras se verifica la autenticación se muestra pantalla de
7     carga
8   if (loading) return <Loader fullScreen />;
9   // Si ya está autenticado se redirige a la página Home
10  if (user) return <Navigate to="/home" />;
11
12  // Formulario para inicio de sesión
13  return (
14    <div className="flex items-center justify-center min-h-screen bg-
15      -gray-100 px-4">
16      <div className="bg-white p-8 rounded shadow-md w-full max-w-sm
17        flex flex-col items-center">
18        <Logo size={172} />
19      </div>
20    </div>
21  );
22 };

```

#### 4. IMPLEMENTACIÓ

```
15     <LoginForm />
16   </div>
17 </div>
18 );
19 };
20
21 // componenst/auth/LoginForm.jsx
22 const LoginForm = () => {
23   const { login } = useAuth(); // Obtener función de inicio de sessi
    ón
24
25   // Añadimos la UI del formulario con las props para el inicio de
    sesión
26   return (
27     <AuthForm
28       title="Iniciar Sesión"
29       fields={[
30         { name: "username", placeholder: "Usuario" },
31         { name: "password", type: "password", placeholder: "Contrase
    ña" },
32       ]}
33       onSubmit={({ username, password }) => login(username, password
    )}
34     />
35   );
36 };
```

La pàgina i funcionalitat principal és la que renderitza `Chatbot.jsx`, Codi A.1 inclòs a l'Annex A.1, gràcies a una variable es van afegint tots els missatges que s'ha de renderitzar. Aquests es van afegint gràcies a `input`, però mentre es carga la pàgina es recupera del backend els darrers missatges del xat que es guarden els primers i precedeixen als que l'usuari i el bot escriuràn. També es destaca la funcionalitat de mostrar un missatge del bot com carregant mentre aquest processa la resposta per millorar la UX i la possibilitat de borrar l'historial i començar un nou xat.

Finalment diverses pàgines comparteixen la necessitat de fer cridades a l'API del mateix mòdul (simplement canvia la ruta i els paràmetres a enviar), per això s'ha creat una funció general per enviar que poden usar tots els arxius per gestionar la comunicació. Això es fa a `apiClient.js`.

Listing 4.17: Codi Frontend: Client API

```
1 // api/apiClient.js
2 export const apiClient = async (
3   endpoint,
4   { method = "GET", body, token, headers = {}, ...customConfig } =
    {}
5 ) => {
6   // Configuración base de la petición HTTP
7   const config = {
8     method, // Método HTTP, por defecto GET
9     headers: {
10       "Content-Type": "application/json",
11       ...(token && { Authorization: 'Bearer ${token}' }), // Añade
        token JWT si está disponible
    }
```



```

12     ...headers,
13   },
14   credentials: "include",
15   ...customConfig, // Permite sobrescribir o añadir
                     configuraciones adicionales
16 };
17
18 // Si se envía un cuerpo en la petición, se convierte a JSON
19 if (body) {
20   config.body = JSON.stringify(body);
21 }
22
23 try {
24   // Se realiza la llamada HTTP a la API
25   const response = await fetch(`${BASE_URL}${endpoint}`, config);
26
27   // Se intenta transformar la respuesta a JSON. Si no es posible,
   se devuelve un objeto vacío.
28   const data = await response.json().catch(() => ({}));
29
30   if (!response.ok) {
31     const message =
32       data?.message || data?.error || "Error en la llamada a la
                                   API";
33     return handleApiError(new Error(message));
34   }
35
36   return { success: true, ...data };
37 } catch (err) {
38   return handleApiError(err);
39 }
40 };

```

### 4.3.2 Backend Node.js

Aquesta és el cervell de l'aplicació, ja que és l'encarregat de gestionar la resta de funcionalitats i mòduls. S'encarrega de què el *frontend* obtingui tota la informació necessària de les BDs i del *backend* de Python. En aquest mòdul es creen *endpoints* per a la comunicació, però cap d'ells renderitza codi com a tal, només exposa funcionalitats que el *frontend*, renderitzat des del client, pot emprar. Tot comença al `index.js` on es gestiona la configuració general del mòdul definint les diferents parts d'aquest com els *middlewares*, les rutes, el port que escolta, etc.

Listing 4.18: Codi Backend Node: Codi Inicial

```

1 // index.js
2 const app = express()
3
4 // Configuración rate limiting global
5 const limiter = rateLimit({
6   windowMs: 1 * 60 * 1000, // 1 minuto
7   max: 1000, // máximo 1000 solicitudes por IP por ventana
8   standardHeaders: true,
9   legacyHeaders: false,
10  message: {

```

#### 4. IMPLEMENTACIÓ

```
11     success: false,
12     message: 'Demasiadas solicitudes. Por favor, inténtalo más tarde
13         .',
14 },
15 })
16
17 // Middlewares de contexto por request (logger, DB, etc.)
18 app.use(attachRequestLogger)
19 app.use(attachRequestDb)
20
21 // Rutas de la API
22 app.get('/', (req, res) =>
23     res.send('Backend activo en ${new Date().toISOString()}')
24 )
25 app.use('/api/example', exampleRoutes)
26 app.use('/api/auth', authRoutes)
27 app.use('/api/users', usersRoutes)
28 app.use('/api/chat', chatRoutes)
29 app.use('/api/dashboard', dashboardRoutes)
30
31 // Middlewares de error (404 y global)
32 app.use(notFoundHandler)
33 app.use(errorHandler)
34
35 app.listen(ENV.PORT, () => {
36     globalLogger.info(
37         {
38             host: ENV.HOST,
39             port: ENV.PORT,
40             environment: ENV.NODE_ENV,
41         },
42         'Servidor backend Node.js iniciado correctamente'
43     )
44 })
```

S'han definit diversos *middlewares*, Codi A.2 inclòs a l'Annex A.2, per afegir funcionalitats interceptant les sol·licituds del client (*frontend*) abans d'executar la lògica principal. Aquest codi intermedi serveix per centralitzar i reutilitzar coses com la connexió amb la BD, el sistema de *logs*, gestionar errors, gestionar cridades a rutes inexistent o assegurar peticions amb autenticació i autorització per rols.

Tant la connexió a la BD de PostgreSQL com a la BD de Redis es representen com objectes amb variables i funcions pròpies. Però per gestionar múltiples connexions i mantenir un control constant s'han creat les *pools*. Tant la connexió a PostgreSQL com a Redis tenen una estructura de codi molt similar, l'únic que canvien són les funcions pròpies de cada tecnologia.

Listing 4.19: Codi Backend Node: Pool de Conexions a les Bases de Dades

```
1 // db/pool.js
2 let pool = createNewPool()
3
4 const RETRY_DELAY_MS = 3000 // 3 segundos
5 const HEALTH_CHECK_INTERVAL_MS = 5 * 60 * 1000 // 5 minutos
6
```

```

7 function createNewPool() { // Pool de conexiones a la base de datos
8   return new Pool({
9     host: ENV.DB.HOST,
10    port: ENV.DB.PORT,
11    user: ENV.DB.USER,
12    password: ENV.DB.PASSWORD,
13    database: ENV.DB.NAME,
14    max: 10,
15    idleTimeoutMillis: 30000,
16    connectionTimeoutMillis: 5000,
17  })
18 }
19
20 // Función periódica para verificar la conexión correcta con la base
  de datos
21 const healthCheck = async () => {
22   try {
23     const client = await pool.connect() // Se obtiene conexión de la
      pool
24     await client.query('SELECT 1') // Se verifica la conexión
25     client.release() // Se cierra la conexión
26   } catch (err) {
27     // Se ejecuta bucle de reconexión
28     if (!isReconnecting) {
29       isReconnecting = true
30
31       while (true) {
32         try {
33           // Cerrar el pool actual antes de intentar uno nuevo
34           await pool.end().catch(() => {}) // Prevenir errores si ya
            está cerrado
35           pool = createNewPool()
36
37           const client = await pool.connect()
38           await client.query('SELECT 1')
39           client.release()
40
41           isReconnecting = false
42           break
43         } catch (reconnectErr) {
44           await wait(RETRY_DELAY_MS)
45         }
46       }
47     }
48   }
49 }
50 setInterval(healthCheck, HEALTH_CHECK_INTERVAL_MS)

```

L'objecte que representa la BD té una funció principal encarregada de comunicar-se directament amb la BD en el llenguatge correcte i assegurant la seguretat de les execucions<sup>3</sup>. També té funcions auxiliars reutilitzables que faciliten l'extracció de dades.

<sup>3</sup>Per exemple, per evitar injeccions SQL s'usa una funció pròpia de la llibreria de PostgreSQL que uneix de manera segura la *query* i els seus paràmetres

Listing 4.20: Codi Backend Node: Base de Dades

```

1 // db/database.js
2 export default class Database {
3   constructor(pool, logger = globalLogger) {
4     this.pool = pool
5     this.logger = logger
6   }
7
8   /**
9    * Ejecuta una query dentro de la base de datos.
10    * @param {String} q = "INSERT INTO users (username, password,
11    *   role) VALUES ($1, $2, $3)"
12    * @param {List} params = [param1, param2, ...]
13    * @returns {Promise<Array>}
14    */
15   async query(q, params = []) {}
16
17   /**
18    * Obtiene la información indicada de los usuarios junto su rol
19    * según los
20    * atributos indicados.
21    * @param {List} attributes ["rol", "nombre", ...]
22    * @param {Object} conditions {nombre: "Juan", username: "juan2924"}
23    * @returns {Promise<Array>}
24    */
25   async getUsersByAttributes(attributes = ['*'], conditions = {}) {}
26
27   /**
28    * Edita uno o varios usuarios indicando los atributos a editar y
29    * los criterios para encontrarlos.
30    * @param {Object} attributes Atributos a actualizar. Ej: {nombre:
31    *   "Juan", rol: "admin", password: "..."}
32    * @param {Object} checkBy Atributos de búsqueda. Ej: {rol: "admin"}
33    * @returns {Promise<Array>}
34    */
35   async updateCommonUsers(attributes = {}, checkBy = {}) {}
36 }

```

Després tenim els arxius de declaració de rutes, aquests se separen per funcionalitat dins la carpeta `api` i a cada un hi podem trobar un o més *endpoints* segons la quantitat de funcions o variants. Per exemple, per les rutes encarregades del xat amb el bot (`/api/chat/talk`) podem trobar la principal (*POST*) per enviar un missatge al bot i rebre una resposta, una per obtenir l'historial de xat d'un usuari (*GET*) i finalment una per eliminar l'historial de xat (*DELETE*).

Listing 4.21: Codi Backend Node: Declaració de Rutes

```

1 // api/chat/routes.js
2 // Aplicar autenticación y autorización a todas las rutas
3 chatRoutes.use(authenticateRequest, authorizeRoles('admin', '
4   developer'))
5
6 /**
7  * POST /api/chat/talk

```

```

7  * API para enviar un mensaje al bot y recibir una respuesta
8  */
9  chatRoutes.post(ENV.API.TALK, talk)
10
11 /**
12  * GET /api/chat/talk
13  * Obtener el historial del chat de un usuario.
14  */
15  chatRoutes.get(ENV.API.TALK, getHistory)
16
17 /**
18  * DELETE /api/chat/talk
19  * Eliminar el historial del chat de un usuario.
20  */
21  chatRoutes.delete(ENV.API.TALK, deleteHistory)

```

Per a cada ruta diferent es defineix una funció concreta encarregada de rebre les consultes, gestionar-les i finalment retornar la resposta. Aquestes funcions s'anomenen controladors i l'estructura és molt similar a tots. Primer s'obtenen totes les variables, tant les que s'han enviat a la consulta com les creades pels *middlewares*. Després es validen que aquestes variables siguin correctes. Si tot va bé podem utilitzar funcions de servei per gestionar la lògica. Finalment s'enviarà la resposta.

Listing 4.22: Codi Backend Node: Controladors

```

1  // controllers/chatController.js
2  /**
3   * POST /api/chat/talk
4   * API para enviar un mensaje al bot y recibir una respuesta
5   */
6  export async function talk(req, res, next) {
7    const logger = req.logger || globalLogger
8    try {
9      const message = req.body?.message
10     const sessionId = req?.user?.sessionId || req.body?.sessionId
11     const userId = req?.user?.id || req.body?.id
12     const rd = req?.redis || globalRedisClient
13     // Validación de parámetros
14     if (!sessionId) {
15       return throwError(CODES.MISSING_TOKEN.CODE, CODES.
16         MISSING_TOKEN.MESSAGE)
17     }
18     if (!userId) {
19       return throwError(CODES.MISSING_TOKEN.CODE, CODES.
20         MISSING_TOKEN.MESSAGE)
21     }
22     validateUserMessage(message)
23
24     // Comunicación con el bot
25     const response = await talkToBot(message, sessionId, rd, userId)
26
27     // Validación de respuesta del bot
28     if (!response || typeof response !== 'string') {
29       return throwError(
30         CODES.BOT_RESPONSE_INVALID.CODE,
31         CODES.BOT_RESPONSE_INVALID.MESSAGE
32       )
33     }
34   } catch (error) {
35     next(error)
36   }
37 }

```

#### 4. IMPLEMENTACIÓ

```
30     )
31   }
32   if (response.length > ENV.LIMIT.MAX_CHAR_BOT_MESSAGE) {
33     response =
34       response.substring(0, ENV.LIMIT.MAX_CHAR_BOT_MESSAGE - 3) +
35       '...'
36   }
37   sendResponse(res, { response })
38 } catch (err) {
39   next(err)
40 }
```

Per dividir i fer més escalable el codi els controladors no executen lògica com a tal, això és la funció dels serveis. Aquests fan coses com recuperar informació de les bases de dades o preparar i enviar els missatges al *backend* de Python. Per exemple, el servei per xerrar amb el bot primer rep el missatge de l'usuari des del controlador, amb l'identificador de l'usuari recupera el seu historial de xat, realitza una petició a l'API de l'altre *backend* amb els paràmetres corresponents i finalment valida la resposta i la retorna al controlador.

Listing 4.23: Codi Backend Node: Serveis

```
1 // services/chatService.js
2 /**
3  * Envía un texto a la API del bot y devuelve la respuesta recibida.
4  *
5  * @param {string} message - Mensaje que se enviará al bot
6  * @param {string} sessionId - ID de la sesión actual
7  * @returns {Promise<string>} - Respuesta del bot
8  * @throws {Error} - Si la API responde con error
9  */
10 export async function talkToBot(message, sessionId, rd =
    globalRedisClient, userId = -1) {
11   let history = {}
12   if (userId > -1) {
13     history = await getChatHistory(rd, userId, 10)
14     if (!history) history = {}
15   }
16
17   const response = await fetch(url + 'talk', {
18     method: 'POST',
19     headers: {
20       'Content-Type': 'application/json',
21       'X-Session-ID': sessionId,
22       'X-Internal-API-Key': ENV.INTERNAL_API_KEY,
23     },
24     body: JSON.stringify({ message, history })
25   })
26   const result = await response.json()
27
28   if (!response.ok || result.status !== 'success') {
29     const errorMessage = result?.errors || 'Error desconocido del
        bot'
30     throw new Error(errorMessage)
31   }
```

```

32     if (userId > -1) {
33         await saveChat(rd, userId, message, result.data)
34     }
35
36     return result.data
37 }

```

### 4.3.3 Backend Python

El *backend* de Python s'ha creat exclusivament per poder usar la llibreria de OpenAI i poder escalar funcionalitats de ML en un futur. Encara així s'ha d'exposar el mòdul per poder comunicar-se amb ell, aquí és on entra la creació d'un *endpoint* amb FastAPI i exposar l'aplicació amb la llibreria Uvicorn. Aquesta darrera permet moltes connexions simultànies i augmenta el rendiment del servidor. Igual que en Node.js, l'arxiu `main.py` executa les configuracions generals del servidor.

Listing 4.24: Codi Backend Python: Codi Inicial

```

1  # main.py
2  app = FastAPI(
3      title="ChatBot Vismel",
4      description="API para interactuar con agente LLM de Vismel",
5      version="1.0.0"
6  )
7
8  app.state.limiter = limiter
9  app.add_middleware(SlowAPIMiddleware)
10
11  setup_routes(app)
12
13  # En caso de error por límite de solicitudes de enviará esta
14  # respuesta
15  @app.exception_handler(RateLimitExceeded)
16  async def rate_limit_handler(request: Request, exc:
17      RateLimitExceeded):
18      return error_response(message="Demasiadas solicitudes. Intenta m
19          ás tarde.", status_code=status.HTTP_429_TOO_MANY_REQUESTS)
20
21  if __name__ == "__main__":
22      uvicorn.run(app, host=HOST, port=PORT)

```

L'*endpoint* que s'ha definit té una estructura molt similar als controladors que s'han vist de Node.js. Aquest rep la consulta amb els paràmetres (missatge de l'usuari, historial, clau privada,...), valida que aquests siguin correctes, crida al servei per processar el missatge i retorna la resposta.

Listing 4.25: Codi Backend Python: Definició de l'API del Bot

```

1  # routes/agent.py
2  """
3  Endpoint para interactuar con el agente. Solo accesible desde el
4  backend autorizado.
5  """
6  @router.post("/talk",
7      response_model=APIResponse,

```

#### 4. IMPLEMENTACIÓ

```
7     name="Conversar con el agente",
8     operation_id="talk_to_agent",
9     description="Recibe una consulta en lenguaje natural, la
    transforma internamente en una consulta SQL, accede a la
    base de datos y devuelve los resultados."
10 )
11 @limiter.limit("20/minute") # Límite de 10 solicitudes por minuto
12 async def talk(
13     request: Request,
14     body: Message,
15     session_id: str = Header(..., alias="X-Session-ID", description=
16     "Identificador único de la sesión"),
17     api_key: str = Header(..., alias="X-Internal-API-Key",
18     description="Clave de autenticación interna")
19 ):
20     if api_key != INTERNAL_API_KEY:
21         return error_response(
22             message="API-Key no válida. No tiene permisos.",
23             status_code=status.HTTP_403_FORBIDDEN
24         )
25     if not session_id.strip():
26         return error_response(
27             message="session_id inválido.",
28             status_code=status.HTTP_400_BAD_REQUEST
29         )
30     try:
31         history = body.history
32         if isinstance(history, dict) or isinstance(history, list):
33             pass
34         else:
35             history = {}
36
37         answer = await process_message(body.message, history, logger
38         )
39
40         if answer is None:
41             return success_response(
42                 message="No se encontraron resultados para la
43                 consulta.",
44                 data=None
45             )
46         return success_response(
47             message="El agente respondió.",
48             data=answer
49         )
```

Per començar a processar el missatge s'usa la funció `process_message`, Codi A.3 inclòs a l'Annex A.3, que aplica la lògica corresponent al flux dissenyat per obtenir la millor resposta. Aquesta funció té l'ajuda de `handle_agent_response`, `handle_extra_query` i `run_query_and_build_answer`, les quals completen el flux i fan el codi més llegible i escalable. Aquestes, aplicant la lògica corresponent explicada a 3.5.3, empren les funcions `get_response`, `get_query_from_previous_data`, `build_answer_from_query` i `build_answer_without_query` del servei de connexió amb OpenAI per passar la informació al bot i rebre una resposta.



Per millorar la latència de l'aplicació i evitar excessives comunicacions entre mòduls s'ha creat un arxiu de connexió amb la BD de PostgreSQL. D'aquesta manera si el bot ha d'executar una consulta SQL per extreure més informació aquesta es pot executar directament sense sortir del mòdul de Python. La seva estructura és molt similar a la comentada al *backend* de Node.js (4.3.2).

Listing 4.26: Codi Backend Python: Base de Dades

```

1  # db/database.py
2  """
3  Objeto que representa una conexion a una DB.
4  """
5  class Database():
6      def __init__(self, user, password, host, port, name, schema):
7          self.schema = schema
8          self.database_url = f"postgresql+asyncpg://{user}:{password}@{host}:{port}/{name}"
9          self.engine = create_async_engine(
10             self.database_url,
11             pool_size=5,
12             max_overflow=10
13         )
14         self.Session = async_sessionmaker(
15             bind=self.engine,
16             expire_on_commit=False,
17             class_=AsyncSession
18         )
19
20     """
21     Prueba la conexión a la base de datos.
22     Devuelve True si se puede conectar, False si falla.
23     """
24     async def test_connection(self, logger=base_logger):
25
26     """
27     Obtiene una descripción completa del esquema de la base de datos
28     ,
29     incluyendo tablas, descripciones de tablas y sus columnas con
30     tipo de dato y descripción.
31     """
32     async def get_schema(self, logger=base_logger):
33
34     """
35     Obten las tablas estáticas de la base de datos.
36     """
37     async def get_static_tables(self, logger=base_logger):
38
39     """
40     Ejecuta una query en la DB y devuelve los resultados.
41     """
42     async def query(self, query: str, logger=base_logger):
43
44     """
45     Cierra la conexión con la DB.
46     """
47     async def close(self, logger=base_logger):

```

Finalment, l'arxiu més important per aconseguir la funcionalitat principal és `openai_llm.py`, Codi A.4 inclòs a l'Annex A.3, que representa la connexió amb l'API d'OpenAI. Aquest té moltes funcions internes que ajuden durant la comunicació (com per exemple per l'ús d'arxius si el context és molt gran), però les més importants són les que formen part del flux de generació de la resposta. Aquestes són molt similars entre elles, la seva única diferència és la informació que gestionen i envien al bot, i el context personalitzat que usen. Primer defineixen la informació dinàmica que s'ha d'afegir al *prompt* (taules estàtiques, esquema de la BD, etc.), després processen les dades i les comparen amb els límits establerts en funció del límit de tokens, si alguna de les dades supera un cert límit de caràcters es guarda en un arxiu temporal i s'envia a l'assistent d'OpenAI, finalment es construeix el *prompt* complet i s'envia la consulta a l'API, depenent de si s'usen arxius pel context o no s'usarà la funcionalitat *chat completions* o la d'*assistant* d'OpenAI (això s'explica millor a la secció de dificultats trobades i com s'han resolt 4.5).

### 4.4 Seguretat en la Implementació

Ja s'han mencionat les distintes mesures que s'han pres per mantenir l'aplicació segura dins les necessitats d'aquesta, però ara es mostrarà de forma més explícita com s'han implementat aquestes mesures de seguretat.

La primera i més important és l'autenticació dins l'aplicació, on com hem vist es fa a través del formulari d'inici de sessió. En aquest formulari l'usuari introdueix un nom d'usuari i la contrasenya corresponent a aquest. La informació és processada per la funció de login del context d'autenticació (`lst:authcontext`) que l'envia a la ruta */api/auth/login*.

El controlador de *login* a Node.js s'encarrega de rebre les dades, validar que són variables correctes i amb el nom d'usuari cerca el registre dins la BD, si es troba un registre és que l'usuari existeix, per tant, passarà a validar la contrasenya. La contrasenya obtinguda de la BD està encriptada, per això, primer es descripta i després es compara amb la rebuda a la consulta, si són iguals podem confirmar que l'inici de sessió és correcte. Finalment, retornem un token d'autenticació amb tota la informació de l'usuari i un temps d'expiració raonable que dona una capa més de seguretat. El token es crea amb una clau personalitzada privada que permet que agents externs a l'aplicació puguin crear o descriptar aquests tokens.

Listing 4.27: Controlador Login de l'API del Backend de Node

```
1 // controllers/authController.js
2 /**
3  * POST /api/auth/login
4  * Controlador encargado de realizar el login comprobando el usuario
5  *   y la contraseña proporcionados.
6  */
7 export async function login(req, res, next) {
8   const logger = req.logger || globalLogger
9   const db = req.db || null
10  try {
11    const username = validateUsername(req.body.username)
```

```

11     const password = req.body.password || ''
12     if (typeof password !== 'string' || password === '') {
13         throwError(CODES.PASSWORD_REQUIRED.CODE, CODES.
14             PASSWORD_REQUIRED.MESSAGE)
15     }
16     validateDBInstance(db)
17
18     const usuarios = await findUserByUsername(db, username.trim())
19     if (usuarios.length !== 1) {
20         throwError(CODES.INVALID_CREDENTIALS.CODE, CODES.
21             INVALID_CREDENTIALS.MESSAGE)
22     }
23
24     const user = usuarios[0]
25     const autenticado = await bcrypt.compare(password, user.password
26         )
27
28     if (!autenticado) {
29         throwError(CODES.INVALID_CREDENTIALS.CODE, CODES.
30             INVALID_CREDENTIALS.MESSAGE)
31     }
32
33     const sessionId = crypto.randomUUID()
34     const token = jwt.sign({id: user?.id, username: username,
35         sessionId, rol: user?.rol, name: user?.nombre,}, ENV.
36         JWT_SECRET, { expiresIn: '2h' })
37
38     res.cookie('token', token, {httpOnly: true, secure: ENV.NODE_ENV
39         === 'production', sameSite: 'Strict', maxAge: 1000 * 60 *
40         60 * 2,})
41
42     sendResponse(res, { token }, 'Login Correcto')
43 } catch (err) {
44     next(err)
45 }
46 }
47 }

```

S'ha mencionat que en extreure la contrasenya de la BD aquesta està encriptada. Les contrasenyes quan es registra un usuari nou dins l'aplicació s'encripten perquè altres usuaris amb accés a la BD o el bot no puguin veure la contrasenya i poder suplantar la seva identitat. Ara mateix això no és un problema, ja que l'únic tipus d'usuari amb accés són els administradors i aquests ja saben la contrasenya perquè són els que creen els usuaris nous. Només s'ha implementat aquesta mesura per evitar que el bot conegui la contrasenya o per si en un futur s'afegeixen altres rols menys importants que l'administrador però amb accés a la BD.

Listing 4.28: Encriptació de Contrasenyes

```

1     const saltRounds = 10
2     const hashedPassword = await bcrypt.hash(password.trim(), saltRounds
3         )
4
5     const newUser = await createNewUser(
6         db,
7         username.trim(),
8         hashedPassword,
9     )

```

## 4. IMPLEMENTACIÓ

---

```
8   rol,  
9   name  
10  )
```

La següent mesura de seguretat adoptada és l'autorització de les funcionalitats. Aquestes depenen del rol de l'usuari on cada un tindrà uns privilegis o uns altres. En el *frontend* per tenir una experiència d'usuari millorada es bloquegen certes rutes segons el rol. Com s'ha mencionat anteriorment, s'empra el component `AuthGuard.jsx` (4.13) per controlar si l'usuari té rol inclòs dins els rols permesos per una ruta concreta. Les rutes protegides i els rols permesos a cada una es defineixen a `AppRouter.jsx` (4.12). Si un usuari navega a aquella ruta serà redirigit a la pàgina (home).

El control d'autenticació i d'autorització també aplica a l'API del *backend* de Node.js, ja que s'exposen rutes que es poden emprar fora del *frontend* per millorar l'escalabilitat. Així si en un futur es crea un altre mòdul que empra aquestes funcionalitats, no serà un perill, ja que implementa la seva pròpia seguretat de manera independent. Això s'aconsegueix amb les funcions de *middleware*: `authenticateRequest` i `authorizeRoles`. Aquests es configuren per executar-se abans de les rutes seleccionades com es veu a 4.21.

La primera, `authenticateRequest` (A.2), rep el token adjunt a la consulta i l'intenta descodificar amb la clau privada, si ho aconsegueix l'autenticació és correcta, sinó significa que l'usuari no està autenticat. La segona, `authorizeRoles` (A.2), es configura a cada ruta amb uns rols autoritzats, quan la consulta passa per aquesta funció obté el rol de l'usuari, a partir de les dades obtingudes durant la descodificació del token en la primera funció, i comprova si aquest rol està dins els rols permesos per aquella ruta.

Per al *backend* de Python la comunicació que permet (de moment) és exclusiva amb l'altre *backend* el qual ja s'encarrega de prendre les mesures de seguretat pertinents. El problema és que al exposar el servidor ens hem d'assegurar que ningú extern pugui fer consultes a l'API, per això s'empra una clau interna privada definida quan es configura l'aplicació i que s'empra per validar que la comunicació prové d'un altre mòdul intern de l'aplicació. Això es pot veure a 4.25.

Finalment, cal mencionar que per a les BDs es té un usuari amb tots els privilegis que serveix per al *backend* de Node.js perquè aquest necessita control absolut, també serveix per a l'encarregat del manteniment, però hi ha un usuari extra que es crea durant l'inici de la BD de PostgreSQL. Aquest només té privilegis de lectura i s'usa per a la connexió a la BD del mòdul de Python, ja que si el bot genera per error una *query* que modifica la BD no es podrà executar correctament.

### 4.5 Dificultats i Solucions Adoptades

L'única dificultat notable que cal mencionar va sorgir durant la implementació del bot amb OpenAI. Després d'estudiar el límit de tokens dels diferents models i fer algunes proves amb l'extracció d'extenses dades de la BD es va trobar que en casos molt especials podria haver-hi errors amb aquests límits per estar construint contextos molt extensos.

Una de les solucions que es va presentar és l'ús del *Retrieval-Augmented Generation* (Generació Augmentada de Recuperació) (RAG) que consisteix en la recuperació d'informació a partir de documents rellevants guardats, normalment, en una BD de vectors. Revisant les eines que oferia l'API d'OpenAI es va trobar el *File Search* ([17]), el qual es diu a la *web* que és « una eina que permet als models cercar entre els teus fitxers informació rellevant abans de generar una resposta ». Això és realment la solució que es necessita, així quan les dades siguin massa llargues simplement s'afegeixen a un fitxer i s'adjunta aquest en la consulta.

Això du amb si una complicació més, ja que aquesta eina no es pot usar amb la interfície que s'havia usat per a les respostes del bot, anomenada *Chat Completions*. L'única manera viable d'usar-ho és a través de l'API d'*Assistants*. Aquesta és una funcionalitat diferent que permet crear un assistent personalitzat a OpenAI amb unes configuracions prèvies, a més pots crear bases de dades on inserir els fitxers i seleccionar-les com informació addicional a usar per a l'assistent.<sup>4</sup>

La primera passa era decidir quan s'havia d'emprar aquesta nova API, per això s'ha dissenyat una fórmula que calcula el límit de caràcters de cada part del *prompt* en funció del límit de tokens que ofereix el model seleccionat. Primer s'ha de tenir en compte és que segons la informació oficial ([18]) 1 token equival a 4 caràcters en anglès, això ho podem extrapolar a l'espanyol. A partir d'això la fórmula diu que el límit de caràcters serà 4 vegades el límit de tokens. Després hi ha 3 parts fonamentals que conformen el *prompt* on a cada una s'estableix una part raonable del límit total de tokens:

1. **Historial de missatges:** L'historial de missatges que el bot recorda es limita a 10 converses (incloent-hi l'actual), per tant, podem dir que es compon per 20 missatges en total. Considerant que les preguntes de l'usuari i les respostes del bot poden ser molt variants i no es pot establir un nombre mitjà de caràcters, es reserven la meitat dels tokens per aquesta part que a la vegada es divideix en els 20 missatges diferents. Per exemple si el límit de caràcters és 800.000 (200.000 tokens) llavors l'historial total seria de 400.000 i cada missatge té un límit de 20.000. Cal mencionar que es lleven 2.000 caràcters a cada missatge per tenir més marge.
2. **Context estàtic:** El context estàtic és molt senzill de mesurar. Actualment sumant els diferents *prompts* estàtics (els que estableixen les regles i instruccions que ha de seguir el bot) tenen en total aproximadament 20.000. Per evitar posar un límit diferent a cada tipus de *prompt* s'estableix aquests 20.000 caràcters com a límit total per així donar més marge a actualitzacions futures.
3. **Context dinàmic:** Per al context dinàmic s'ha considerat la resta de tokens. És a dir, seria en la meitat dels caràcters totals (perquè l'altra meitat els usa l'historial) menys els 20.000 caràcters considerats pel *prompt* fix.

---

<sup>4</sup>La latència de l'API d'*Assistants* és major que la de *Chat Completions*, per tant, es continuarà usant aquesta darrera mentre no sigui necessari l'ús dels fitxers per poder millorar la UX

Així el càlcul quedaria com:

$$C_{\max} = T \cdot 4$$

$$C_h = \frac{C_{\max}}{2}$$

$$C_m = \frac{C_h}{20} - 2000$$

$$C_e = 20000$$

$$C_d = \frac{C_{\max}}{2} - C_e$$

Llegenda:

- $T$ : Límit de tokens de context del model seleccionat
- $C_{\max}$ : Nombre màxim de caràcters per al context total
- $C_h$ : Nombre màxim de caràcters per l'història
- $C_m$ : Nombre màxim de caràcters per missatge
- $C_e$ : Nombre màxim de caràcters per context estàtic
- $C_d$ : Nombre màxim de caràcters per context dinàmic

Després, des de la *web* d'OpenAI s'ha creat manualment un assistent i una BD de vectors, tots dos tenen identificadors propis que es guarden com a variables d'entorn en el mòdul de Python. Aquests identificadors ens serviran per seleccionar-los durant l'inici del servei `openai_llm.py`, així i tot, si no es poden seleccionar es crearan uns temporals per aquella sessió. També a l'inici, per evitar crear i guardar fitxers constantment, es recuperen els fitxers que ja estan guardats a la BD de vectors.

L'assistant funciona amb fils de conversa, per a cada missatge que envia l'usuari s'inicia un fil on s'insereix l'història de missatges, es passa el *prompt* i es genera una resposta. Aquest fil està actiu durant tot el flux i és usat per totes les funcions es comunicaran amb el bot. Una vegada es genera la resposta final aquest fil es tanca.

Per decidir quan s'ha d'obrir un fil amb l'assistant s'usa el límit de caràcters de context dinàmic mencionat abans. Les dades es guarden dins un diccionari amb el nom de les dades com a clau i les dades en si com a valor. Aquest diccionari es passa a la funció `_process_prompts` la qual s'encarrega de comprovar si la suma de caràcters entre les diferents dades supera el límit total. Si es considera que unes dades han de ser processades en un fitxer llavors el crea, l'envia a la BD de vectors de l'assistant (o obté

l'identificador si ja hi ha un fitxer amb el mateix nom que s'havia guardat anteriorment<sup>5</sup>) i al diccionari es posa com a valor *'In the file with id: id\_fitxer'* indicant a quin fitxer es troben les dades corresponents. Tot això ho fa de forma òptima, ja que només transforma les que siguin completament necessàries.

Listing 4.29: Funció de Processament de les Dades per usar-les al Context

```

1 # services/openai_llm.py
2 """
3 Procesa múltiples fragmentos de datos, asegurando que el total no
4 exceda el máximo.
5 """
6 async def processprompts(self, prompt_parts, max_length):
7     processed = {name: await self._process_prompt_part(name, content,
8         max_length) for name, content in prompt_parts}
9
10    total_length = sum(len(p) for p in processed.values())
11
12    # Si aún combinados sobrepasan el límite, dividir más
13    if total_length > max_length:
14        for name, original_content in prompt_parts:
15            if not processed[name].startswith("In the file"):
16                file = await self._create_file(original_content, name=name)
17                processed[name] = f"In the file with id: {file.id}"
18                total_length = sum(len(p) for p in processed.values())
19                if total_length <= max_length:
20                    break
21    return processed

```

Quan s'ha processat aquest diccionari de dades es construeix el *prompt*, és a dir, el *prompt* estàtic usat per la funció corresponent és omplit amb les dades dinàmiques corresponents usant el diccionari, això vol dir que si alguna de les dades s'ha guardat dins un fitxer en lloc d'escriure directament les dades al *prompt* s'escriu la frase *'In the file with id: id\_fitxer'* indicant al bot on pot trobar aquestes dades.

Si alguna de les dades ha estat guardada dins un fitxer i enviada a la BD de vectors s'iniciarà el processament de la resposta a través del fil de l'API d'*Assistants*, sinó es continuarà com de costum amb *Chat Completions*.

Listing 4.30: Exemple de Preparació Prèvia a una Interacció amb el Model d'OpenAI

```

1 # services/openai_llm.py
2 """
3 El agente genera una respuesta a partir del mensaje del usuario y de
4 los datos obtenidos de la DB.
5 """
6 async def build_answer_from_query(self, result, message, db_schema:
7     = "NO DATA", static_tables = {}, sql_query = "NO DATA", history:
8     = []):
9     # Eliminamos el fichero result por si hay restos

```

<sup>5</sup>Els fitxers que es guarden a la BD de vectors es queden guardats allà usant-los com a memòria *cache*, d'aquesta manera en processaments posteriors la latència és menor, ja que no s'han de tornar a guardar. Només es realitza amb les dades dinàmiques més 'estàtiques' de la BD com les taules que no solen canviar o l'esquema complet, la resta s'esborren després de ser usades.

#### 4. IMPLEMENTACIÓ

```
7   file_result = self.openai_files.get_file_by_name(name="result.json")
8   if file_result:
9       await self._delete_file(id=file_result.id, name=file_result.name)
10
11  prompt_parts = [("db_schema.txt", str(db_schema)), ("result.json",
12  result)]
13  for table_name, table_data in static_tables.items():
14      prompt_parts.append((f"{table_name}.json", table_data))
15
16  # Procesamos los datos para evitar que se exceda el límite de
17  tokens
18  processed_prompts = await self._process_prompts(prompt_parts,
19  MAX_CHAR_DATA)
20  db_schema_prompt = processed_prompts["db_schema.txt"]
21  result_prompt = processed_prompts["result.json"]
22  static_tables_prompt = "\n---\n".join(f"{k}\n{v}" for k, v in
23  processed_prompts.items() if k not in {"db_schema.txt", "
24  result.json"})
25
26  # Construimos el prompt con los datos dinámicos
27  prompt = build_prompt(PROMPTS["answer_with_data"], {"db_schema":
28  db_schema_prompt, "static_tables": static_tables_prompt, "
29  sql_query": sql_query, "result": result_prompt, "lang": self.
30  lang, "limit": self.limit})
31
32  # Si alguno de los datos se ha guardado en un fichero se llamara
33  al assistant, sino será una consulta normal
34  if any(value.startswith("In the file") for value in
35  processed_prompts.values()):
36      await self._continue_thread()
37      await self._insert_history_messages_in_thread(history=history)
38      await self._add_message_to_thread(message=message, role=user)
39      res = await self._run_thread(prompt=prompt)
40  else:
41      messages = build_messages(prompt=prompt, message=message,
42      history=history)
43      res = await self._call_openai(messages=messages)
44
45  # Eliminamos el fichero result por si hay restos
46  file_result = self.openai_files.get_file_by_name(name="result.json")
47  if file_result:
48      await self._delete_file(id=file_result.id, name=file_result.name)
49
50  res_json = json.loads(res) if res else {}
51  response = res_json.get("response", "")
52
53  return str(response)
```

Finalment, per al control dels missatges enviats per l'usuari i de les respostes generades pel bot es fa una validació d'aquests dins el controlador de la ruta del *backend* de Node.js encarregat de la conversa amb el bot, comparant-los amb els caràcters màxims establerts per cada missatge individual. Si el missatge de l'usuari és massa llarg es



retorna un error, mentre que si la resposta del bot és massa llarga aquesta es retalla fins a complir el límit. Per a millorar la UX, dins el *frontend*, l'*input* del missatge de l'usuari no li permet escriure més del límit establert.



## PROVES I VALIDACIÓ

### 5.1 Introducció a les Proves

L'objectiu de les proves és comprovar si el xatbot és capaç de respondre de manera coherent i amb dades correctes, tant en preguntes relacionades amb l'empresa i la seva BD, com en preguntes generals sense cap relació amb Vismel.

En particular, el bot ha de ser capaç de generar les sentències SQL necessàries, fins i tot en consultes complexes, i d'interpretar els resultats de forma alineada amb la intencionalitat de l'usuari.

La finalitat d'aquest procés és identificar possibles millores i ajustos tant en el flux de la conversa com en els *prompts* utilitzats durant les interaccions. En cas que el marge de millora sigui inexistent o resulti massa costós, es definirà les limitacions pròpies del xatbot i establir quines estratègies poden emprar-se.

### 5.2 Estratègia de les Proves

Per tal d'avaluar el rendiment del xatbot, s'ha definit una metodologia basada en l'ús d'un conjunt de preguntes. Aquest conjunt es divideix en dos grups principals:

- **Preguntes dissenyades per l'autor del projecte:** les preguntes es basen en el coneixement del disseny i implementació tant de la BD del flux de generació de la resposta. Aquestes inclouen tant consultes simples com consultes complexes sobre la BD, així com preguntes alienes a l'àmbit de l'empresa. L'objectiu d'aquest grup és comprovar la capacitat del xatbot per generar sentències SQL correctes i per gestionar situacions en què la informació sol·licitada no està disponible a la base de dades.

- **Preguntes reals d'un usuari de Vismel:** permeten validar el comportament del sistema en un escenari més proper a l'ús real dins de l'entorn empresarial.

Per a cada pregunta es va registrar la resposta generada pel xatbot i posteriorment es va avaluar segons tres categories de qualitat:

1. **Correcte:** la resposta és coherent i resol correctament la consulta plantejada.
2. **Incompleta:** la resposta aporta informació parcial, però no resol del tot la consulta.
3. **Incorrecte:** la resposta és incorrecta o no té relació amb la pregunta.

Aquesta estratègia permet obtenir una visió general de la fiabilitat del sistema, tant en l'execució de consultes sobre la base de dades com en la gestió de preguntes generals.

### 5.3 Resultats de les Proves Realitzades

A continuació es mostren exemples dels resultats obtinguts a les diferents proves realitzades al xatbot. Cada targeta inclou la pregunta plantejada, la resposta donada pel sistema i l'avaluació final (Correcte, Incompleta o Incorrecte). El llistat complet de preguntes i respostes es pot trobar en l'Annex A.4.

#### Pregunta: ¿Qué tipos de familia de artículos existen?

*Resposta del bot:*

PRODUCTOS DE CAFE  
RECAMBIOS MAQUINARIA  
OTROS  
HORAS Y DESPLAZAMIENTOS  
VENTA RECAUDACION DE MAQUINAS  
BEBIDAS

*Avaluació:* Correcte

**Pregunta: ¿Cómo funciona el protocolo HTTP?**

*Resposta del bot:*

El protocolo HTTP (Hypertext Transfer Protocol) es el sistema básico de comunicación en la web. Funciona bajo un modelo cliente-servidor, en el que un cliente (por ejemplo, un navegador) envía una petición HTTP a un servidor, y este responde con los datos solicitados (normalmente páginas web, imágenes, documentos, etc.). Principales características del funcionamiento de HTTP:

Es un protocolo sin estado: cada petición es independiente y no guarda información de las anteriores.

Utiliza los métodos principales: GET (solicita un recurso), POST (envía datos), PUT (modifica un recurso), DELETE (elimina un recurso), entre otros.

Funciona habitualmente sobre el puerto 80 para HTTP y el 443 para HTTPS (versión segura).

Las respuestas llevan un código de estado, como 200 (OK), 404 (no encontrado) o 500 (error interno).

Resumen del proceso:

El cliente abre una conexión al servidor y envía una petición HTTP.

El servidor procesa la petición, accede al recurso solicitado y envía una respuesta HTTP.

El cliente recibe la respuesta y la muestra o la procesa según corresponda.

HTTP es fundamental para la transmisión de información en la web.

*Avaluació:* Correcte

**Pregunta: ¿Cuáles son los 5 artículos más utilizados en salidas durante 2024 y cuánto stock queda de cada uno?**

*Resposta del bot:*

Los 5 artículos más utilizados en salidas durante el año 2024 son:

CODO MACHO 8/6 1/8 S/R con 49 unidades salidas y un stock actual de 45 unidades.

CONECTOR POTENCIA 4 VIAS con 36 unidades salidas y un stock actual de 4 unidades.

BANDEJA MARRO CAFMATI con 35 unidades salidas y un stock actual de 4 unidades.

CONTENEDOR RECTANGULAR 115/460 con 33 unidades salidas y un stock actual de 3 unidades.

RECOGEGOTAS CAFMATI con 31 unidades salidas y un stock actual de 5 unidades.

Estos datos reflejan el volumen de uso y la cantidad disponible de cada uno en inventario.

*Avaluació:* Correcte

**Pregunta: ¿Cuál será la recaudación estimada de la máquina 42 en la semana 32 de 2025? ¿Por qué llegas a esa conclusión?**

*Resposta del bot:*

No disponemos de datos de recaudación de la máquina 42 para la semana 32 de los últimos 9 años. Los últimos valores disponibles son: 11,50 € en 2015, 36,60 € en 2013, 48,00 € en 2012, 65,00 € en 2011 y 122,00 € en 2010, con una tendencia claramente descendente. No hay registros desde 2015, lo que puede indicar que la máquina ya no está operativa, que apenas se recauda o simplemente que no se registró.

Por lo tanto, la recaudación estimada para la semana 32 de 2025 sería muy baja o incluso nula, siguiendo la tendencia histórica y la ausencia reciente de recaudaciones.

Motivo de la estimación:

El descenso continuado de ingresos en la serie histórica.

Ausencia total de recaudaciones en los años recientes.

La incertidumbre porque no hay datos actuales.

Si se espera un cambio (reactivación, nueva ubicación, etc.), sería imprescindible información adicional para ajustar la previsión.

*Avaluació:* Correcte

**Pregunta de l'usuari: Compara la recaudación del cliente taller vismel de 2013 y 2014**

*Resposta del bot:*

La recaudación total del cliente TALLER VISMEI fue:

En 2013: 68.020,95 €

En 2014: 67.944,21 €

Se observa que la recaudación en 2014 fue ligeramente menor que en 2013, con una diferencia de 76,74 € menos, lo que indica una estabilidad en los ingresos entre ambos años.

*Avaluació:* Incompleta (Comparació molt bàsica, alomillor amb una millor consulta es pot treure més informació.)

**Pregunta de l'usuari: ¿Cuales son las 5 máquinas que han hecho menor número de servicios en 2014?**

*Resposta del bot:*

Las 5 máquinas que realizaron el menor número de servicios en 2014, según la lectura máxima registrada de su contador, son las siguientes:

Máquina con ID 25: 111 servicios

Máquina con ID 267: 148 servicios

Máquina con ID 61: 246 servicios

Máquina con ID 158: 265 servicios

Máquina con ID 342: 637 servicios

Estas cifras corresponden al total acumulado de servicios realizados por cada máquina durante ese año.

*Avaluació:* Correcte

## 5.4 Conclusions i Limitacions

Les proves realitzades mostren que el xatbot ha estat capaç de respondre de manera correcta a totes les preguntes plantejades, tant les relacionades amb la BD de l'empresa com les de caràcter general. Això mostra que el sistema és funcional i que la integració amb la BD permet generar consultes SQL vàlides, interpretar-ne els resultats i retornar respostes coherents amb la intencionalitat de l'usuari.

Tot i aquests resultats positius, l'anàlisi també ha evidenciat algunes limitacions. S'ha observat que, per obtenir respostes completament correctes, sovint ha estat necessari formular les preguntes de manera més detallada. En casos en què la consulta era més oberta o ambigua, el xatbot tendia a donar respostes parcials o interpretacions alternatives que, si bé no eren incorrectes, no sempre coincidien amb la necessitat real de l'usuari. Aquesta dependència de la precisió en la formulació és una limitació general dels models de llenguatge actuals, que mostren dificultats a l'hora de gestionar ambigüitats i contextos poc definits.

També s'ha identificat que la qualitat de les respostes depèn en gran mesura de la disponibilitat i la qualitat de les dades existents a la base de dades. En preguntes que requereixen informació predictiva o que parteixen de registres incomplets, el xatbot només pot oferir respostes aproximades, la qual cosa pot limitar la seva utilitat.

Es pot concloure que el sistema desenvolupat és una eina vàlida per donar suport a la consulta d'informació de l'empresa. Així mateix, presenta limitacions derivades tant de la naturalesa dels LLM com de les dades disponibles.





## CONCLUSIONS I TREBALL FUTUR

El xatbot ha demostrat ser capaç de generar consultes SQL correctes i tornar respostes coherents en la majoria dels casos, incloent-hi preguntes més complexes. La integració amb la BD s'ha mostrat viable, i ha permès extreure informació rellevant de manera natural i oferint un suport útil com a assistent virtual.

L'anàlisi també revela una sèrie de limitacions que han de ser considerades. En primer lloc, el sistema depèn en gran part de com formula l'usuari la pregunta. Quan la consulta és poc precisa, el model tendeix a produir resultats parcials o interpretacions alternatives que no sempre s'ajusten a la necessitat real. Aquest comportament reflecteix una limitació dels LLMs actuals.

La qualitat de les respostes predictives o basades en tendències es veu condicionada per la disponibilitat de les dades de la BD. En absència d'informació suficient, el bot només pot oferir estimacions aproximades.

Un altre aspecte rellevant és la dependència del sistema respecte del model de llenguatge escollit. En aquest treball s'ha fet servir un model que ofereix una capacitat suficient per respondre de manera adequada a les consultes plantejades. No obstant això, no es pot garantir que altres models tinguin el mateix rendiment o que s'adaptin correctament a l'aplicació desenvolupada.

A més, s'ha observat que, encara que en general les consultes SQL generades són correctes, hi ha una certa variabilitat en la formulació de sentències. En la majoria dels cas, aquesta diferència no afecta el resultat final, però en algunes poques ocasions s'han generat consultes incorrectes o que fallaven internament.

En conclusió, el desenvolupament del xatbot i tota l'estructura que l'envolta ha donat resultats positius, ja que els problemes més notables que es poden trobar es donen per limitacions de la tecnologia actual. Aquestes limitacions, especialment la dependència

del model i la variabilitat en la generació de consultes SQL, posen de manifest que l'eina requereix manteniment i millores contínues per assolir un nivell òptim de fiabilitat en entorns productius. Això vol dir que hi ha molt marge de millora i, a mesura que els LLMs vagin avançant, el xatbot també ho farà si es duu a terme el manteniment necessari.

El *software* creat dona peu a moltes millores de cara a tenir una eina completa i funcional que centralitzi totes les funcions necessàries per a l'empresa Vismel i els seus usuaris. Les pròximes passes inclouen noves funcionalitats empresarials, com ara la gestió de l'inventari, el seguiment d'avaries i torns de manteniment, el monitoratge de recaptacions o la digitalització de factures. També es poden desenvolupar solucions addicionals, com l'automatització de rutes de manteniment setmanals i la incorporació de mòduls de predicció i anàlisi de tendències sobre la BD, mitjançant models d'aprenentatge profund.

En paral·lel, també cal tenir en compte les millores tècniques del propi xatbot. Entre elles destaquen la incorporació (si es viable) del model GPT-5 recentment llançat i les seves variants, i l'adaptació a les noves APIs d'OpenAI davant l'obsolescència de l'API d'*Assistants* incorporant alguna solució equivalent i millorada per part de l'empresa.

Finalment, la implementació de tot el servei de la connexió i conversa amb el model d'OpenAI usant la seva llibreria de Python de manera escalable dona peu a crear noves aplicacions desenvolupant solucions específiques i especialitzades amb aquesta funcionalitat de xatbot o similar. Aquest tipus d'aplicacions que implementen IA de maneres creatives són el futur de tot *software* i una gran millora de l'escena informàtica actual.



## ANNEXOS

### A.1 Codi Frontend

Part del codi font del *frontend*.

Listing A.1: Codi Frontend: Xat amb el Bot

```
1 // pages/Chatbot.jsx
2 const handleSend = async (text) => {
3   const userMessage = { message: text, main: true };
4
5   // Se añade el mensaje del usuario junto con el loading del bot
6   addMessages(userMessage, loadingMessage);
7   try {
8     // Llamada a la API encargada de enviar el mensaje al bot
9     const { success, message, response } = await talkToBot(
10       token,           // Token de autenticación
11       text,            // Mensaje enviado
12       user.sessionID // Identificador único de la sesión
13     );
14     if (success && response) {
15       replaceLoadingWithResponse(response);
16     } else {
17       // Si hubo fallo en la respuesta, se elimina el mensaje de
18       // carga y se notifica el error
19       removeLoading();
20       toast.error(message);
21     }
22   } catch (error) {
23     removeLoading();
24     toast.error("Error al comunicarse con el bot");
25   }
26 };
27
28 const deleteChat = () => {
29   // Se guarda el historial actual de mensajes en memoria temporal
```

```

29   const messages_cache = messages;
30
31   toast.promise(
32     (async () => {
33       setMessages([]); // Se limpia la conversación en la interfaz
34       // Llamada a la API encargada de borrar el historial
35       const { success, message } = await deleteHistory(token);
36       if (!success) {
37         setMessages(messages_cache);
38         throw new Error(message || "No se pudo eliminar el historial");
39       }
40       return success;
41     })(),
42     {
43       loading: "Eliminando historial...",
44       success: "¡Historial borrado correctamente!",
45       error: (err) => `Error al borrar: ${err.message}`,
46     }
47   );
48 };
49
50 // Hook que se ejecuta al cargar el componente o cuando cambia el
51 // token de autenticación
52 useEffect(() => {
53   const fetchHistory = async () => {
54     try {
55       // Llamada a la API encargada de obtener el historial
56       const { success, response, message } = await getHistory(token);
57       if (success) {
58         // Se invierte el orden para mostrar de lo más antiguo a lo
59         // más reciente
60         const history = [];
61         response
62           .slice()
63           .reverse()
64           .forEach((item) => {
65             if (item?.user) history.push({ message: item.user, main: true });
66             if (item?.bot) history.push({ message: item.bot, main: false });
67           });
68         // Se añaden los mensajes recuperados
69         setMessages((prev) => [...prev, ...history]);
70       } else {
71         // Si no hay historial, se inicializa con un mensaje de
72         // bienvenida del bot
73         setMessages((prev) => [
74           ...prev,
75           { message: "¡Hola! ¿En qué puedo ayudarte hoy?", main: false },
76         ]);
77       }
78     } catch (error) {
79       setMessages((prev) => [
80         ...prev,

```

```

78     { message: "¡Hola! ¿En qué puedo ayudarte hoy?", main: false
79         },
80     ];
81 };
82
83 if (token) {
84     fetchHistory();
85 }
86 }, [token]);

```

## A.2 Codi Backend Node

Part del codi font del *backend* de Node.js.

Listing A.2: Codi Backend Node: Middlewares

```

1 // middleware/attachRequestDb.js
2 const attachRequestDb = async (req, res, next) => {
3     try {
4         req.db = new Database(pool, req.logger) // Se crea la conexión
5           con base de datos principal
6         req.redis = globalRedisClient // Se crea la conexión con cache
7         await req.db.testConnection() // Se verifica la conexión
8         next()
9     } catch (error) {
10         next(error)
11     }
12 }
13
14 // middleware/authenticateRequest.js
15 const authenticateRequest = (req, res, next) => {
16     // Se obtiene el token enviado con la petición
17     const token = req.cookies?.token || req.headers.authorization?.
18       split(' ')[1]
19     try {
20         const decoded = jwt.verify(token, ENV.JWT_SECRET) //
21           Decodificamos token
22         if (!decoded || !decoded.username) {
23             return next(
24               throwError(CODES.INVALID_TOKEN.CODE, CODES.INVALID_TOKEN.
25                 MESSAGE)
26             )
27         }
28
29         // Si es correcto rellenamos la info del usuario
30         req.user = {
31             id: decoded.id,
32             username: decoded.username,
33             rol: decoded.rol,
34             sessionID: decoded.sessionID,
35         }
36         next()
37     } catch (err) {
38         return next(
39             throwError(

```

```

36         CODES.TOKEN_EXPIRED_OR_INVALID.CODE,
37         CODES.TOKEN_EXPIRED_OR_INVALID.MESSAGE
38     )
39 )
40 }
41 }
42
43 // middleware/authorizeRoles.js
44 const authorizeRoles = (...allowedRoles) => {
45     return (req, res, next) => {
46         // Se obtiene el rol del usuario y se comprueba con la lista de
47         // roles autorizados
48         const userRole = req.user?.rol
49         if (!allowedRoles.includes(userRole)) {
50             return next(throwError(CODES.FORBIDDEN.CODE, CODES.FORBIDDEN.
51                 MESSAGE))
52         }
53         next()
54     }
55 }
56
57 // middleware/attachRequestLogger.js
58 const attachRequestLogger = (req, res, next) => {
59     // Se obtiene el token enviado con la petición
60     const token = req.cookies?.token || req.headers.authorization?.
61         split(' ')[1]
62     try {
63         const decoded = jwt.verify(token, ENV.JWT_SECRET) //
64             Decodificamos token
65         req.logger = globalLogger.child({ // Se crea objeto logger
66             con la info de la petición
67             user: decoded?.user || 'anonymous',
68             sessionId: decoded?.sessionId || null,
69             requestId: uuidv4(),
70         })
71         req.user = decoded?.user
72     } catch (err) {
73         req.logger = globalLogger.child({ // Se crea objeto logger
74             anónimo
75             anonymous: true,
76             requestId: uuidv4(),
77         })
78     }
79 }

```

### A.3 Codi Backend Python

Part del codi font del *backend* de Python.

Listing A.3: Codi Backend Python: Flux de la Resposta del Bot

```

1 # services/process_message.py
2 async def process_message(message, history, logger):
3     """
4     Procesa un mensaje entrante.

```

```

5      Intenta obtener respuesta desde la DB mediante queries. Si falla
      , responde sin datos.
6      """
7      try:
8          if agent and db:
9              # 1. Verificar conexión a la base de datos
10             if not await db.test_connection(logger):
11                 return await agent.build_answer_without_query(
12                     message, history)
13             # 2. Obtener esquema de la base de datos
14             db_schema = await db.get_schema(logger)
15             static_tables = await db.get_static_tables(logger)
16             if not db_schema:
17                 return await agent.build_answer_without_query(
18                     message, static_tables, history)
19             # 3. Obtener respuesta del agente
20             raw_response = await agent.get_response(message,
21                 db_schema, static_tables, history)
22             if not raw_response:
23                 return await agent.build_answer_without_query(
24                     message, db_schema, static_tables, history)
25             agent_response = json.loads(raw_response)
26
27             return await handle_agent_response(agent_response,
28                 message, db_schema, static_tables, history, logger)
29
30 async def handle_agent_response(agent_response, message, db_schema,
31     static_tables, history, logger):
32     """
33     Maneja la respuesta del agente y construye la respuesta final.
34     """
35     try:
36         response = agent_response.get("response", "")
37         query = agent_response.get("sql_query", "")
38         extra_sql_query = agent_response.get("extra_sql_query", "")
39
40         if query:
41             return await run_query_and_build_answer(query, message,
42                 db_schema, static_tables, history, logger)
43         elif extra_sql_query:
44             return await handle_extra_query(extra_sql_query, message
45                 , db_schema, static_tables, history, logger)
46         elif response:
47             return response
48
49         return await agent.build_answer_without_query(message,
50             db_schema, static_tables, history) if agent else None
51
52 async def handle_extra_query(extra_query, message, db_schema,
53     static_tables, history, logger):
54     """
55     Maneja el flujo cuando se necesita una query adicional antes de
56     construir la final.
57     """
58     try:
59         if agent and db:

```

```

50         results = await db.query(extra_query, logger)
51         if not results:
52             return await agent.build_answer_without_query(
53                 message, db_schema, static_tables, sql_query,
54                 history)
55
56         res2 = await agent.get_query_from_previous_data(message,
57             db_schema, static_tables, extra_query, results,
58             history)
59         if not res2:
60             return await agent.build_answer_without_query(
61                 message, db_schema, static_tables, history)
62
63         agent_response2 = json.loads(res2)
64         response = agent_response2.get("response", "")
65         query = agent_response2.get("sql_query", "")
66
67         if query:
68             return await run_query_and_build_answer(query,
69                 message, db_schema, static_tables, history,
70                 logger)
71         elif response:
72             return response
73
74         return await agent.build_answer_without_query(message,
75             db_schema, static_tables, history)
76
77 async def run_query_and_build_answer(query, message, db_schema,
78     static_tables, history, logger):
79     """
80     Ejecuta una query y construye una respuesta a partir de sus
81     resultados.
82     """
83     try:
84         if agent and db:
85             results = await db.query(query, logger)
86             if results:
87                 return await agent.build_answer_from_query(results,
88                     message, db_schema, static_tables, query,
89                     history)
90             else:
91                 return await agent.build_answer_without_query(
92                     message, db_schema, static_tables, query,
93                     history)

```

Listing A.4: Codi Backend Python: Comunicació amb OpenAI API

```

1 # services/openai_llm.py
2 """
3 El agente genera una query SQL para extraer los datos necesarios, o
4 directamente, genera una respuesta según el mensaje del usuario.
5 """
6 async def get_response(self, message, db_schema = "NO DATA",
7     static_tables = {}, history = []):
8     prompt_parts = [("db_schema.txt", str(db_schema))]
9     for table_name, table_data in static_tables.items():
10         prompt_parts.append((f"{table_name}.json", table_data))

```



```

10     processed_prompts = await self._process_prompts(prompt_parts,
11             MAX_CHAR_DATA)
12     db_schema_prompt = processed_prompts["db_schema.txt"]
13     static_tables_prompt = "\n---\n".join(f"{k}\n{v}" for k, v in
14         processed_prompts.items() if k != "db_schema.txt")
15
16     prompt = build_prompt(PROMPTS["get_response"], {"db_schema":
17         db_schema_prompt, "static_tables": static_tables_prompt, "
18         lang": self.lang, "limit": self.limit})
19
20     # Si alguno de los datos se ha guardado en un fichero se llamara
21     # al assistant, sino será una consulta normal
22     if any(value.startswith("In the file") for value in
23         processed_prompts.values()):
24         await self.continue_thread()
25         await self._insert_history_messages_in_thread(history=
26             history)
27         await self._add_message_to_thread(message=message, role=user
28             )
29         return await self._run_thread(prompt=prompt)
30     else:
31         messages = build_messages(prompt=prompt, message=message,
32             history=history)
33         return await self._call_openai(messages=messages)
34
35 """
36 El agente genera una query SQL para extraer los datos necesarios a
37 partir de una query previa debido a la indeterminación de los
38 datos como podría ser un nombre.
39 """
40
41 async def get_query_from_previous_data(self, message, db_schema = "
42 NO DATA", static_tables = {}, previous_sql = "NO DATA",
43 previous_result = "NO DATA", history = []):
44     # Eliminamos el fichero previous_result por si hay restos
45     file_previous_results = self.openai_files.get_file_by_name(name=
46         "previous_result.json")
47     if file_previous_results:
48         await self._delete_file(id=file_previous_results.id, name=
49             file_previous_results.name)
50
51     prompt_parts = [("db_schema.txt", str(db_schema)), ("
52         previous_result.json", previous_result)]
53     for table_name, table_data in static_tables.items():
54         prompt_parts.append((f"{table_name}.json", table_data))
55
56     # Aqui decidimos si guardar el contenido en un fichero o pasarlo
57     # directamente en el prompt
58     processed_prompts = await self._process_prompts(prompt_parts,
59         MAX_CHAR_DATA)
60
61     db_schema_prompt = processed_prompts["db_schema.txt"]
62     previous_result_prompt = processed_prompts["previous_result.json
63         "]
64     static_tables_prompt = "\n---\n".join(f"{k}\n{v}" for k, v in
65         processed_prompts.items() if k not in {"db_schema.txt", "
66         previous_result.json"})

```

```

46     prompt = build_prompt(PROMPTS["get_query_from_previous_data"], {
47         "db_schema": db_schema_prompt, "static_tables":
48         static_tables_prompt, "previous_sql": previous_sql, "
49         previous_result": previous_result_prompt, "lang": self.lang,
50         "limit": self.limit})
51
52     # Si alguno de los datos se ha guardado en un fichero se llamara
53     # al assistant, sino será una consulta normal
54     if any(value.startswith("In the file") for value in
55         processed_prompts.values()):
56         await self.continue_thread()
57         await self._insert_history_messages_in_thread(history=
58             history)
59         await self._add_message_to_thread(message=message, role=user
60             )
61         res = await self._run_thread(prompt=prompt)
62     else:
63         messages = build_messages(prompt=prompt, message=message,
64             history=history)
65         res = await self._call_openai(messages=messages)
66
67     # Eliminamos el fichero de previous_result
68     file_previous_results = self.openai_files.get_file_by_name(name=
69         "previous_result.json")
70     if file_previous_results:
71         await self._delete_file(id=file_previous_results.id, name=
72             file_previous_results.name)
73
74     return res
75
76 """
77 El agente genera una respuesta a partir del mensaje del usuario y de
78 los datos obtenidos de la DB.
79 """
80
81 async def build_answer_from_query(self, result, message, db_schema =
82     "NO DATA", static_tables = {}, sql_query = "NO DATA", history =
83     []):
84     # Eliminamos el fichero result por si hay restos
85     file_result = self.openai_files.get_file_by_name(name="result.
86         json")
87     if file_result:
88         await self._delete_file(id=file_result.id, name=file_result.
89             name)
90
91     prompt_parts: List[Tuple[str, str | dict | list[dict]]] = [(
92         "db_schema.txt", str(db_schema)), ("result.json", result)]
93     for table_name, table_data in static_tables.items():
94         prompt_parts.append((f"{table_name}.json", table_data))
95
96     processed_prompts = await self._process_prompts(prompt_parts,
97         MAX_CHAR_DATA)
98     db_schema_prompt = processed_prompts["db_schema.txt"]
99     result_prompt = processed_prompts["result.json"]
100     static_tables_prompt = "\n---\n".join(f"{k}\n{v}" for k, v in
101         processed_prompts.items() if k not in {"db_schema.txt", "
102             result.json"})

```

```

83     prompt = build_prompt(PROMPTS["answer_with_data"], {"db_schema":
84         db_schema_prompt, "static_tables": static_tables_prompt, "
            sql_query": sql_query, "result": result_prompt, "lang": self
            .lang, "limit": self.limit})

85
86     # Si alguno de los datos se ha guardado en un fichero se llamara
            al assistant, sino será una consulta normal
87     if any(value.startswith("In the file") for value in
            processed_prompts.values()):
88         await self.continue_thread()
89         await self._insert_history_messages_in_thread(history=
            history)
90         await self._add_message_to_thread(message=message, role=user
            )
91         res = await self._run_thread(prompt=prompt)
92     else:
93         messages = build_messages(prompt=prompt, message=message,
            history=history)
94         res = await self._call_openai(messages=messages)
95
96     # Eliminamos el fichero result por si hay restos
97     file_result = self.openai_files.get_file_by_name(name="result.
            json")
98     if file_result:
99         await self._delete_file(id=file_result.id, name=file_result.
            name)

100
101     res_json = json.loads(res) if res else {}
102     response = res_json.get("response", "")
103
104     return str(response)
105
106 """
107 El agente genera una respuesta a partir del mensaje del usuario sin
            tener los datos de la DB.
108 """
109 async def build_answer_without_query(self, message, db_schema,
            static_tables, sql_query, history):
110     prompt_parts = [("db_schema.txt", str(db_schema))]
111     for table_name, table_data in static_tables.items():
112         prompt_parts.append((f"{table_name}.json", table_data))
113
114     processed_prompts = await self._process_prompts(prompt_parts,
            MAX_CHAR_DATA)
115     db_schema_prompt = processed_prompts["db_schema.txt"]
116     static_tables_prompt = "\n---\n".join(f"{k}\n{v}" for k, v in
            processed_prompts.items() if k != "db_schema.txt")
117
118     prompt = build_prompt(PROMPTS["answer_without_data"], {"
            db_schema": db_schema_prompt, "static_tables":
            static_tables_prompt, "sql_query": sql_query, "lang": self.
            lang, "limit": self.limit})
119
120     # Si alguno de los datos se ha guardado en un fichero se llamara
            al assistant, sino será una consulta normal
121     if any(value.startswith("In the file") for value in

```

```
122         processed_prompts.values()):
123             await self.continue_thread()
124             await self._insert_history_messages_in_thread(history=
                history)
125             await self._add_message_to_thread(message=message, role=user
                )
126             res = await self._run_thread(prompt=prompt)
127         else:
128             messages = build_messages(prompt=prompt, message=message,
                history=history)
129             res = await self._call_openai(messages=messages)
130
131         res_json = json.loads(res) if res else {}
132         response = res_json.get("response", "")
133
134         return str(response)
```

## A.4 Llistat de Preguntes i Respostes del Xatbot

Preguntes y respuestas empraades durant les proves del xatbot i dividides entre: preguntes formulades per l'autor del TFG i preguntes formulades per un usuari de l'empresa. Cada targeta inclou la pregunta plantejada, la resposta donada pel sistema i l'avaluació final (Correcte, Incompleta o Incorrecte).

### A.4.1 Preguntes Generades per l'Autor

#### Simplex

##### Pregunta: ¿Qué tipos de familia de artículos existen?

*Resposta del bot:*

PRODUCTOS DE CAFE  
RECAMBIOS MAQUINARIA  
OTROS  
HORAS Y DESPLAZAMIENTOS  
VENTA RECAUDACION DE MAQUINAS  
BEBIDAS

*Avaluació:* Correcte

##### Pregunta: ¿Cuáles son los roles disponibles en el sistema?

*Resposta del bot:*

Administrador  
Técnico  
Cliente  
Operario

Estos son los roles disponibles en el sistema.

*Avaluació:* Correcte

**Pregunta: ¿Qué frecuencias de revisión hay y cada cuántos meses corresponden?**

*Resposta del bot:*

Mensual: cada 1 mes

Trimestral: cada 3 meses

Semestral: cada 6 meses

*Avaluació:* Correcte

**Pregunta: Muestra la información del usuario Toni Campi**

*Resposta del bot:*

La búsqueda para 'Toni Campi' arrojó coincidencias, pero ninguna es exacta. ¿Te refieres a 'TONYI CAMPILLO' (username: TONYI\_CAMPILLO, similitud: 0,44) o a 'TONI AMER' (username: TONI\_AMER, similitud: 0,31)? Por favor, indícanos cuál de estos usuarios es el correcto para mostrar su información.

*Avaluació:* Correcte

**Pregunta: Necesito datos de la máquina con código 4**

*Resposta del bot:*

Modelo: PALMA B

Tipo: REFRESCOS

Conector: Sí

Tarjetero: No

Billetero: No

Orain/Visa: No

Usuario predeterminado: TONI RAMIS

Cliente: RESIDENCIA SAN JUAN

Frecuencia de revisión: Semestral (cada 6 meses)

Coste de adquisición: 3,00 €

Fecha inicio de amortización: 01/01/2002

Observaciones: nº serie 10038-597

*Avaluació:* Correcte

**Pregunta: ¿Cuántos artículos 'Café' hay en stock y cuál es su precio medio?**

*Resposta del bot:*

Hay un total de 1471 artículos de la familia 'Café' en stock.

El precio medio de estos artículos es de 8,29 € por unidad.

*Avaluació:* Correcte

**Pregunta: ¿Cuál fue el importe recaudado total la semana 12 de 2011 para la máquina 42?**

*Resposta del bot:*

El importe total recaudado en la semana 12 del año 2011 para la máquina 42 fue de 13,20 €.

*Avaluació:* Correcte

**Pregunta: Dame la lista de averías pendientes de alta prioridad**

*Resposta del bot:*

13/08/2025: Máquina 7 - Billetero bloqueado  
10/08/2025: Máquina 11 - Revisión necesaria  
07/08/2025: Máquina 6 - Batería baja  
04/08/2025: Máquina 9 - Orain Visa no funciona  
01/08/2025: Máquina 4 - Fallo en conector  
23/04/2025: Máquina 194 - Pantalla táctil sin respuesta  
09/04/2025: Máquina 13 - Pantalla táctil sin respuesta

*Avaluació:* Correcte

**Pregunta: ¿Cuándo se reparó por última vez la máquina 75?**

*Resposta del bot:*

La máquina 75 fue reparada por última vez el 17 de julio de 2024.

*Avaluació:* Correcte

**Pregunta: ¿Cuál es el coste total de la familia 'Bebidas'?**

*Resposta del bot:*

El coste total acumulado de todos los artículos pertenecientes a la familia 'Bebidas' es de 14,03 €.

*Avaluació:* Correcte

**Pregunta: ¿Qué clientes tienen la máquina con modelo prisma y dónde están ubicados?**

*Resposta del bot:*

BAR IDEAL está ubicada en INCA.  
HOTELES GLOBALES tiene varias máquinas ubicadas en 1 PALMA NOVA (BALEARES).  
BASKET SOLLER está ubicada en SOLLER.  
CALA MARÇAL CLUB HOTEL está ubicada en PORTO COLOM (BALEARES).

*Avaluació:* Correcte

**Pregunta: Muestra nombre y contacto de la persona encargada de las averías con prioridad 2 en la máquina 208**

*Resposta del bot:*

Las averías con prioridad 2 de la máquina 208 han sido gestionadas por:  
JOSE ANTONIO MURIEL (usuario: JOSE\_ANTONIO\_MURIEL)  
TOMEU FIOL (usuario: TOMEU\_FIOL)

Para el contacto directo, consulta el listado interno de empleados o solicita información adicional si necesitas sus datos de contacto (como teléfono o email).

*Avaluació:* Correcte

**Pregunta: Lista de revisiones realizadas en abril de 2022 con tipo limpieza**

*Resposta del bot:*

3 de abril de 2022: Máquina 122, técnico: TECNICOS, comentario: Botón atascado  
4 de abril de 2022: Máquina 91, técnico: JOSE ANTONIO MURIEL, sin comentarios  
8 de abril de 2022: Máquina 41, técnico: TONI RAMIS, sin comentarios  
9 de abril de 2022: Máquina 70, técnico: TERESA, comentario: Filtro sucio  
9 de abril de 2022: Máquina 137, técnico: TECNICOS, comentario: Botón atascado  
10 de abril de 2022: Máquina 192, técnico: TECNICOS, sin comentarios  
16 de abril de 2022: Máquina 102, técnico: TOFOL JOFRE, comentario: Ruido inusual  
26 de abril de 2022: Máquina 90, técnico: TOMEU FIOL, sin comentarios  
27 de abril de 2022: Máquina 184, técnico: TERESA, comentario: Ruido inusual  
30 de abril de 2022: Máquina 136, técnico: TOFOL JOFRE, comentario: Ruido inusual

*Avaluació:* Correcte

**Pregunta: ¿Cuál ha sido la recaudación media mensual de la máquina 4 en el segundo semestre de 2007?**

*Resposta del bot:*

La recaudación media mensual de la máquina 4 durante el segundo semestre de 2007 fue de 26,18 €.

*Avaluació:* Correcte

**Pregunta: ¿Cuántas averías se reportaron en mayo de 2024?**

*Resposta del bot:*

En mayo de 2024 se reportaron un total de 5 averías.

*Avaluació:* Correcte

**Generals (Fora de Context)**

**Pregunta: ¿Cuál es la capital de España?**

*Resposta del bot:*

La capital de España es Madrid.

*Avaluació:* Correcte

**Pregunta: ¿Cómo funciona el protocolo HTTP?**

*Resposta del bot:*

El protocolo HTTP (Hypertext Transfer Protocol) es el sistema básico de comunicación en la web. Funciona bajo un modelo cliente-servidor, en el que un cliente (por ejemplo, un navegador) envía una petición HTTP a un servidor, y este responde con los datos solicitados (normalmente páginas web, imágenes, documentos, etc.). Principales características del funcionamiento de HTTP:

Es un protocolo sin estado: cada petición es independiente y no guarda información de las anteriores.

Utiliza los métodos principales: GET (solicita un recurso), POST (envía datos), PUT (modifica un recurso), DELETE (elimina un recurso), entre otros.

Funciona habitualmente sobre el puerto 80 para HTTP y el 443 para HTTPS (versión segura).

Las respuestas llevan un código de estado, como 200 (OK), 404 (no encontrado) o 500 (error interno).

**Resumen del proceso:**

El cliente abre una conexión al servidor y envía una petición HTTP.

El servidor procesa la petición, accede al recurso solicitado y envía una respuesta HTTP.

El cliente recibe la respuesta y la muestra o la procesa según corresponda.

HTTP es fundamental para la transmisión de información en la web.

*Avaluació:* Correcte



### Complexes

**Pregunta: ¿Cuáles son los 5 artículos más utilizados en salidas durante 2024 y cuánto stock queda de cada uno?**

*Resposta del bot:*

Los 5 artículos más utilizados en salidas durante el año 2024 son:

CODO MACHO 8/6 1/8 S/R con 49 unidades salidas y un stock actual de 45 unidades.

CONECTOR POTENCIA 4 VIAS con 36 unidades salidas y un stock actual de 4 unidades.

BANDEJA MARRO CAFMATI con 35 unidades salidas y un stock actual de 4 unidades.

CONTENEDOR RECTANGULAR 115/460 con 33 unidades salidas y un stock actual de 3 unidades.

RECOGEGOTAS CAFMATI con 31 unidades salidas y un stock actual de 5 unidades.

Estos datos reflejan el volumen de uso y la cantidad disponible de cada uno en inventario.

*Avaluació:* Correcte

**Pregunta: ¿Qué usuario ha realizado más revisiones de tipo limpieza en el último año?**

*Resposta del bot:*

El usuario que ha realizado más revisiones de tipo limpieza en el último año es TONI AMER, con un total de 15 revisiones de este tipo.

*Avaluació:* Correcte

**Pregunta: ¿Qué máquina ha generado más ingresos promedio por semana en 2015?**

*Resposta del bot:*

La máquina que ha generado más ingresos promedio por semana en 2015 es la máquina con ID 272. Su ingreso medio semanal fue de 226,81 € durante ese año.

*Avaluació:* Correcte

**Pregunta: Muestra los 3 clientes con más máquinas en propiedad**

*Resposta del bot:*

Los 3 clientes con más máquinas en propiedad son:

TALLER VISMEL, con 39 máquinas

HOTELES GLOBALES, con 35 máquinas

CONCESION CALVIA, con 15 máquinas

*Avaluació:* Correcte

**Pregunta: ¿Cuáles son las 10 averías con comentarios más largos sin completar?**

*Resposta del bot:*

Buscar en averia con completado = false, usar LENGTH(comentarios) o similar, ordenar por longitud descendente.

Las 10 averías no completadas con los comentarios más largos actualmente son:

Pantalla táctil sin respuesta

Pantalla táctil sin respuesta (aparece dos veces en la lista, indicando que se repite el problema)

Tarjetero necesita limpieza

Error en lectura de tickets

Orain Visa no funciona

Problema con billeteiro

Error de comunicación

Pantalla no responde

Problema en software

Billetero bloqueado

Estas averías tienen comentarios que van desde 19 hasta 29 caracteres, reflejando incidencias principalmente en la pantalla táctil, problemas con el billeteiro y el sistema de pago.

*Avaluació:* Correcte

**Pregunta: ¿Qué población tiene mayor número de máquinas instaladas?**

*Resposta del bot:*

La población con mayor número de máquinas instaladas es INCA, con un total de 46 máquinas.

*Avaluació:* Correcte

**Pregunta: ¿Qué máquina tiene el coste más alto y qué cliente la posee?**

*Resposta del bot:*

La máquina con el coste más alto es la de código 64, con un coste de 901,54 €. Esta máquina pertenece al cliente JOAN XXIII.

*Avaluació:* Correcte

**Pregunta: ¿Qué artículos tienen un margen negativo (coste > precio) y están en stock?**

*Resposta del bot:*

Los siguientes artículos tienen un margen negativo (el coste es superior al precio de venta) y actualmente están en stock:

FASTON HEMBRA 6,35: precio 0,08 € | coste 0,28 € | stock 792 uds  
 FUNDA FASTON MIXTO 6,35: precio 0,07 € | coste 0,10 € | stock 789 uds  
 PULSADOR MEMBRANA GRANDE PCI: precio 0,24 € | coste 0,36 € | stock 653 uds  
 MICRO RODILLO: precio 5,50 € | coste 9,80 € | stock 649 uds  
 FUSIBLE 1A 5 X 20 MM: precio 0,17 € | coste 0,24 € | stock 595 uds  
 MANGUERA SILICONA 4 X 8: precio 2,31 € | coste 2,44 € | stock 486 uds  
 TERMINAL PASO UNITARIO 3,96: precio 0,07 € | coste 0,12 € | stock 443 uds  
 FUSIBLE 4A 5 X 20: precio 0,13 € | coste 0,18 € | stock 389 uds  
 MANGUERA NYLON 2 X 4: precio 1,16 € | coste 1,17 € | stock 323 uds  
 MICRO PATILLA: precio 5,50 € | coste 5,75 € | stock 294 uds  
 JUNTA TORICA VITON 7x1,5 CÁMARA: precio 0,50 € | coste 0,57 € | stock 294 uds  
 MANGUERA SILICONA 8 X 12: precio 3,18 € | coste 4,09 € | stock 272 uds  
 NESTEA LIMON LATA 33CL (24U): precio 0,57 € | coste 0,95 € | stock 264 uds  
 TERMINAL FASTON HEMBRA 2,8: precio 0,07 € | coste 0,16 € | stock 257 uds  
 BICONO 4/2: precio 0,33 € | coste 0,40 € | stock 195 uds  
 PULSADOR AUTOZUMO: precio 1,20 € | coste 1,94 € | stock 183 uds  
 BICONO 6/4: precio 0,33 € | coste 0,35 € | stock 155 uds  
 JUNTA EMBOLO PEQUEÑO 30,3x3: precio 1,65 € | coste 1,66 € | stock 153 uds  
 COCACOLA ZERO SIN CAFEINA (24U): precio 0,57 € | coste 0,79 € | stock 144 uds  
 PULSADOR CROMADO ROJO IMPOR.: precio 2,31 € | coste 2,98 € | stock 143 uds  
 JUNTA TÓRICA VITON 72,62X3,53: precio 2,50 € | coste 3,50 € | stock 143 uds  
 CODO SALIDA PRODUCTO AUTO 10mm NEGRO: precio 0,90 € | coste 0,96 € | stock 131 uds

...

NOTA: La lista es extensa. Si necesitas información detallada de un artículo concreto o de una familia, ¡dímelo!

*Avaluació:* Correcte

**Pregunta: ¿Qué tipo de máquina tiene en promedio mayor número de averías?**

*Resposta del bot:*

El tipo de máquina con mayor número de averías registradas es CAFÉ, con un total de 142 averías.

Le siguen las máquinas de REFRESCOS con 96 averías y las de SNACKS con 48 averías.

Por tanto, las máquinas de tipo CAFÉ son, en promedio, las que presentan más incidencias de averías en Vismel.

*Avaluació:* Correcte

**Pregunta: ¿Qué usuario gestionó más averías en julio de 2021?**

*Resposta del bot:*

El usuario que gestionó más averías en julio de 2021 fue TOMEU FIOL, con un total de 3 averías registradas durante ese mes.

*Avaluació:* Correcte

**Predicció i Tendència**

**Pregunta: ¿Cuál será la recaudación estimada de la máquina 42 en la semana 32 de 2025? ¿Por qué llegas a esa conclusión?**

*Resposta del bot:*

No disponemos de datos de recaudación de la máquina 42 para la semana 32 de los últimos 9 años. Los últimos valores disponibles son: 11,50 € en 2015, 36,60 € en 2013, 48,00 € en 2012, 65,00 € en 2011 y 122,00 € en 2010, con una tendencia claramente descendente. No hay registros desde 2015, lo que puede indicar que la máquina ya no está operativa, que apenas se recauda o simplemente que no se registró.

Por lo tanto, la recaudación estimada para la semana 32 de 2025 sería muy baja o incluso nula, siguiendo la tendencia histórica y la ausencia reciente de recaudaciones.

Motivo de la estimación:

El descenso continuado de ingresos en la serie histórica.

Ausencia total de recaudaciones en los años recientes.

La incertidumbre porque no hay datos actuales.

Si se espera un cambio (reactivación, nueva ubicación, etc.), sería imprescindible información adicional para ajustar la previsión.

*Avaluació:* Correcte

**Pregunta:** Segun el registro histórico de entradas y salidas de artículos del inventario de años anteriores. ¿Se esperaba una caída en el stock de artículos de la familia 'Bebidas' en mayo de 2024?

*Resposta del bot:*

Analizando el registro histórico de entradas y salidas para artículos de la familia 'Bebidas', se observa lo siguiente para mayo de 2024:

En mayo de 2024 hubo solo salidas (5 unidades) y ninguna entrada, resultando en un saldo neto negativo de -5 unidades.

Los meses anteriores muestran un patrón variable:

Abril de 2024: saldo neto positivo (+4), con solo entradas y sin salidas.

Marzo de 2024: saldo neto positivo (+2), solo entradas.

Junio de 2024: saldo neto negativo (-10), solo salidas.

En los años anteriores, no hay un patrón claro de entradas antes de mayo seguido de una caída solo en ese mes. Por ejemplo, mayo de 2023 fue saldo positivo (+5); mayo de 2022, saldo positivo (+13); mayo de 2021, saldo negativo (-2).

No se aprecia una tendencia clara en la que el saldo neto de mayo caiga habitualmente ni que sea previsible una caída como la ocurrida en 2024. El saldo negativo en mayo de 2024 parece más una excepción puntual y no una repetición estacional basada en el histórico.

En resumen:

No se esperaba una caída en el stock de 'Bebidas' en mayo de 2024 según el registro de años anteriores, ya que no hay un patrón estacional definido para ese mes.

*Avaluació:* Correcte

**Pregunta: ¿Qué cliente podría requerir una revisión preventiva este mes basándose en la frecuencia de revisión de sus máquinas?**

*Respuesta del bot:*

Este mes, según las frecuencias de revisión registradas y las fechas de inicio de amortización, los siguientes clientes tienen máquinas que podrían requerir una revisión preventiva:

FRANZL

Máquina: 8 (Modelo: COFFEMAR250)

Frecuencia: Semestral

Fecha de inicio de amortización: 01/08/2018

HOTELES GLOBALES

Máquina: 45 (Modelo: PRISMA)

Frecuencia: Semestral

Fecha de inicio de amortización: 10/02/2000

BONGRUP INCA

Máquina: 26 (Modelo: ARTIC 510)

Frecuencia: Semestral

Fecha de inicio de amortización: 11/02/2002

ROMAR GIMNASIO

Máquina: 110 (Modelo: COFFEMAR230)

Frecuencia: Semestral

Fecha de inicio de amortización: 01/08/2020

MARTE HOTEL

Máquina: 129 (Modelo: COFFEMAR500)

Frecuencia: Trimestral

Fecha de inicio de amortización: 11/05/2009

Estas máquinas, según su frecuencia y el ciclo actual, cumplen los requisitos para revisión preventiva en este mes.

*Avaluació:* Correcte

**Pregunta:** En mayo de 2010, ¿Qué máquina debería ser retirada por baja rentabilidad basada en sus ingresos en los últimos 12 meses?

*Resposta del bot:*

La máquina con menor rentabilidad basada en los ingresos totales de los últimos 12 meses (de mayo de 2009 a mayo de 2010) es la siguiente:

Modelo: PALMA B

Tipo: REFRESCOS

Cliente: BONGRUP POLLENÇA

Código interno: 165

Ingresos totales en los últimos 12 meses: 6,00 €

Esta máquina es la candidata más clara para ser retirada por baja rentabilidad, ya que sus ingresos en el periodo analizado son extremadamente bajos.

*Avaluació:* Correcte

#### A.4.2 Preguntas Formuladas per un Usuari de l'Empresa

**Pregunta de l'usuari:** ¿Cual es la recaudación total del mes de enero del 2014?

*Resposta del bot:*

La recaudación total del mes de enero de 2014 fue de 31.171,05 €

*Avaluació:* Correcte

**Pregunta de l'usuari:** Compara la recaudación del cliente taller vismel de 2013 y 2014

*Resposta del bot:*

La recaudación total del cliente TALLER VISMEL fue:

En 2013: 68.020,95 €

En 2014: 67.944,21 €

Se observa que la recaudación en 2014 fue ligeramente menor que en 2013, con una diferencia de 76,74 € menos, lo que indica una estabilidad en los ingresos entre ambos años.

*Avaluació:* Incompleta (Comparació molt bàsica, alomillor amb una millor consulta es pot treure més informació.)

**Pregunta de l'usuari: ¿Cuales son las 5 máquinas que han hecho menor número de servicios en 2014?**

*Resposta del bot:*

Las 5 máquinas que realizaron el menor número de servicios en 2014, según la lectura máxima registrada de su contador, son las siguientes:

Máquina con ID 25: 111 servicios

Máquina con ID 267: 148 servicios

Máquina con ID 61: 246 servicios

Máquina con ID 158: 265 servicios

Máquina con ID 342: 637 servicios

Estas cifras corresponden al total acumulado de servicios realizados por cada máquina durante ese año.

*Avaluació:* Correcte

**Pregunta de l'usuari: Listado de maquinas asignadas al empleado Toni Ramis**

*Resposta del bot:*

Máquinas asignadas a Toni Ramis:

Código 2, Modelo COFFEMAR250, Tipo CAFÉ

Código 4, Modelo PALMA B, Tipo REFRESCOS

Código 6, Modelo PRISMA, Tipo CAFÉ

Código 7, Modelo PRISMA, Tipo CAFÉ

Código 14, Modelo COFFEMAR250, Tipo CAFÉ

Código 19, Modelo VISION, Tipo SNACKS

Código 21, Modelo COFFEMAR500, Tipo CAFÉ

Código 25, Modelo COFFEMAR250, Tipo CAFÉ

Código 31, Modelo NOVA PEQ. FRIO, Tipo SNACK/BEBIDA

Código 34, Modelo COFFEMAR300BLUE, Tipo CAFÉ

Código 35, Modelo COFFEMAR250, Tipo CAFÉ

Código 37, Modelo VISION, Tipo SNACKS

Código 41, Modelo COFFEMAR500, Tipo CAFÉ

Código 48, Modelo SECONDOMA AUTOMATICA, Tipo CAFÉ

Código 50, Modelo COFFEMAR250, Tipo CAFÉ

Código 57, Modelo FAS(sin lec.), Tipo SNACKS

...

*Avaluació:* Correcte

**Pregunta de l'usuari: ¿En qué mes del año, de media, se venden más artículos SERPENTIN AGUA D-ZUMO?**

*Resposta del bot:*

El mes del año en el que, de media, salen más artículos del artículo SERPENTIN AGUA D-ZUMO es junio, con un total máximo de 43 unidades salidas en ese mes registrado.

*Avaluació:* Correcte



**Pregunta de l'usuari: ¿Cuántos SERPENTÍN AGUA D-ZUMO saca cada empleado en cada mes?**

*Resposta del bot:*

La media mensual de unidades de 'SERPENTIN AGUA D-ZUMO' sacadas por empleado es la siguiente:

JOSE ANTONIO MURIEL: 7,5 unidades/mes

TERESA: 19 unidades/mes

TOFOL JOFRE: 12,33 unidades/mes

TONI AMER: 11 unidades/mes

El resto de empleados no ha registrado salida de este artículo o la media es 0.

*Avaluació:* Incompleta (Es refereix a cada mes individual. Millorable amb una consulta més específica.)



## GLOSSARI

**ChatGPT** Model conversacional desenvolupat per l'empresa OpenAI [5]. xi

**Claude** [claude.ai](https://claude.ai) [19]. xi

**Debian** Distribució de Linux gratuïta i de codi obert [20]. 9

**Docker** [docker.com](https://docker.com) [21]. 2, 47, 48, 52

**Express** [expressjs.com](https://expressjs.com) [22]. 10

**FastAPI** [fastapi.tiangolo.com](https://fastapi.tiangolo.com) [23]. 10, 63

**Firewall** Dispositiu de seguretat de xarxa dissenyat per supervisar, filtrar i controlar el trànsit de xarxa entrant i sortint segons regles de seguretat predeterminades. L'objectiu principal d'un tallafocs és establir una barrera entre una xarxa interna de confiança i xarxes externes no fiables. 7

**Gemini** [gemini.google.com](https://gemini.google.com) [24]. xi

**Git** Sistema de control de versions distribuït que fa un seguiment de les versions dels fitxers.. 9

**IBM Watson Assistant** [watsonx-assistant](https://watsonx-assistant) [1]. 5

**JavaScript** [Wiki JavaScript](https://wiki.javascript) [25]. ix, xi, 9

**LangChain** [langchain.com](https://langchain.com) [4]. 5

**LLaMA** [llama.com](https://llama.com) [7]. 6

**Node.js** [nodejs.org](https://nodejs.org) [26]. xi, 3, 4, 10, 15, 16, 29, 30, 63, 65, 66, 68, 72, 85

**Ollama** [ollama.org](https://ollama.org) [6]. 6

**Open Source (Codi Obert)** Creacions on el codi font, el disseny o el contingut es posa a disposició pública, cosa que permet la modificació, redistribució i ús per part de qualsevol persona.. 3, 9

**OpenAI** [openai.com](https://openai.com) [5]. xi, 2, 3, 5, 10–12, 15, 16, 30–32, 43, 44, 48, 63, 64, 66, 68–70, 82, 107

**PostgreSQL** [postgresql.org](https://postgresql.org) [27]. 9, 15, 44, 45, 48, 58, 59, 65, 68

**prompt** Entrada o instrucció que un usuari proporciona a un model d'intel·ligència artificial per obtenir una resposta o executar una acció específica.. 31, 32, 34, 37, 39, 40

**Python** [python.org](https://python.org) [28]. xi, 3, 4, 10, 15, 30, 57, 62, 63, 65, 68, 70, 82, 86

**React** [react.dev](https://react.dev) [29]. 3, 9, 41

**Redis** [redis.io](https://redis.io) [30]. xi, 3, 9, 15, 16, 44, 45, 48, 58

**SWE-Bench** Dissenyat per avaluar la capacitat dels LLM en la resolució de tasques de programació. Consisteix en un conjunt de problemes extrets de projectes de programari de codi obert. L'objectiu és mesurar en quina mesura un model pot generar solucions que resolguin els errors o implementin les funcionalitats sol·licitades, simulant així un escenari de desenvolupament professional.. 12

**Uvicorn** [uvicorn.org](https://uvicorn.org). 63

**Vite** [vite.dev](https://vite.dev). 3, 9, 41

**VSCode** Editor de codi més usat en tot el món gràcies a les àmplies funcionalitats natives integrades i a la gran quantitat d'extensions que permeten un disseny, desenvolupament i proves de tot un projecte completament d'inici a fi [31]. 9

**Web App** Aplicació a la que s'accedeix utilitzant com a client un navegador web usant el protocol TCP/IP, ja sigui a través d'internet o d'una intranet. xi, 1, 7, 52

## BIBLIOGRAFIA

- [1] IBM, "Ibm watsonx assistant," *IBM*, 2025. [Online]. Available: <https://www.ibm.com/products/watsonx-assistant> 2.2, A.4.2
- [2] A. Liu, X. Hu, L. Wen, and P. S. Yu, "A comprehensive evaluation of chatgpt's zero-shot text-to-sql capability," *arXiv*, 2023. [Online]. Available: <https://arxiv.org/abs/2303.13547> 2.2
- [3] M. Joseph, H. Raj, A. Yadav, and A. Sharma, "Askyourdb: An end-to-end system for querying and visualizing relational databases using natural language," *arXiv*, 2022. [Online]. Available: <https://arxiv.org/abs/2210.08532> 2.2
- [4] H. Chase. (2025) Langchain. [Online]. Available: <https://www.langchain.com/langchain> 2.2, A.4.2
- [5] OpenAI. (2025) Open models by openai. [Online]. Available: <https://openai.com/open-models/> 2.2, A.4.2
- [6] Ollama. (2025) Ollama. [Online]. Available: <https://ollama.org/> 2.2, A.4.2
- [7] Meta. (2025) Llama 2. [Online]. Available: <https://www.llama.com/docs/overview/> 2.2, A.4.2
- [8] OpenAI. (2025) Openai api models. [Online]. Available: <https://platform.openai.com/docs/models/> 2.5.3
- [9] A. A. A. F. AI). (2025) Ultimate comparison of gpt-4.1 vs gpt-4o: Which one should you use? [Online]. Available: <https://felloai.com/2025/04/ultimate-comparison-of-gpt-4-1-vs-gpt-4o-which-one-should-you-use/> 2
- [10] Leucopsis, "How gpt-4.1 compares to gpt-4o," *Medium*, 2025. [Online]. Available: <https://medium.com/@leucopsis/how-gpt-4-1-compares-to-gpt-4o-5e7d9a52d113> 3, 4
- [11] Wikipedia, "Gpt-4o," *Wikipedia*, 2025. [Online]. Available: <https://en.wikipedia.org/wiki/GPT-4o> 3
- [12] F. Labs, "Gpt-4o vs gpt-4.1: A detailed comparison," *F22 Labs*, 2025. [Online]. Available: <https://www.f22labs.com/blogs/gpt-4o-vs-gpt-4-1-a-detailed-comparison> 4

- [13] ScrumLaunch, “Openai gpt models differences: Gpt-4o vs gpt-4.1,” *ScrumLaunch*, 2025. [Online]. Available: <https://www.scrumlaunch.com/blog/openai-gpt-models-differences> 4
- [14] Reuters, “Openai launches new gpt-4.1 models with improved coding, long context comprehension,” *Reuters*, 2025. [Online]. Available: <https://www.reuters.com/technology/artificial-intelligence/openai-launches-new-gpt-41-models-with-improved-coding-long-context-2025-04-14> 4
- [15] OpenAI. (2025) Gpt-4.1 models documentation. [Online]. Available: <https://platform.openai.com/docs/models/gpt-4.1> 4
- [16] J. A. Olivares. (2024) Cómo conectar tu base de datos a un chatbot con inteligencia artificial. [Online]. Available: <https://www.youtube.com/watch?v=3KVqlIF0zZw> 2
- [17] OpenAI. (2025) Openai api tools: File search. [Online]. Available: <https://platform.openai.com/docs/guides/tools-file-search> 4.5
- [18] ——. (2025) What are tokens and how to count them? [Online]. Available: <https://help.openai.com/en/articles/4936856-what-are-tokens-and-how-to-count-them> 4.5
- [19] Anthropic. (2025) Claude. [Online]. Available: <https://claude.ai> A.4.2
- [20] Debian Project. (2025) Debian. [Online]. Available: <https://www.debian.org/> A.4.2
- [21] Docker Inc. (2025) Docker. [Online]. Available: <https://www.docker.com/> A.4.2
- [22] Expressjs.com. (2025) Express. [Online]. Available: <https://expressjs.com/> A.4.2
- [23] Tiangolo. (2025) Fastapi. [Online]. Available: <https://fastapi.tiangolo.com> A.4.2
- [24] Google LLC. (2025) Gemini. [Online]. Available: <https://gemini.google.com/app> A.4.2
- [25] ECMA International. (2025) Javascript. [Online]. Available: <https://www.ecma-international.org/> A.4.2
- [26] Node.js Foundation. (2025) Node.js. [Online]. Available: <https://nodejs.org/> A.4.2
- [27] PostgreSQL Global Development Group. (2025) Postgresql. [Online]. Available: <https://www.postgresql.org/> A.4.2
- [28] Python Software Foundation. (2025) Python. [Online]. Available: <https://www.python.org/> A.4.2
- [29] Meta Platforms, Inc. (2025) React. [Online]. Available: <https://react.dev> A.4.2
- [30] Redis Inc. (2025) Redis. [Online]. Available: <https://redis.io/> A.4.2
- [31] Microsoft Corporation. (2025) Visual studio code. [Online]. Available: <https://code.visualstudio.com/> A.4.2