Oriol Poblet Roca

Comparador d'Assignatures d'Erasmus amb Intel·ligència Artificial

TREBALL DE FI DE GRAU

dirigit per Jordi Duch Gavaldà

Grau d'Enginyeria Informàtica

UNIVERSITAT ROVIRA I VIRGILI

Tarragona

2025

Resum.

Aquest projecte neix per abordar la problemàtica recurrent i manual de la convalidació d'assignatures en estades Erasmus, un procés que genera incertesa als estudiants i una càrrega administrativa als coordinadors. La solució proposada és un projecte d'un comparador automatitzat que, mitjançant intel·ligència artificial, analitza i compara els continguts de les guies docents. Aquest programa es basa en un backend amb FastAPI i Python que, usant els models Llama3 i nomic-embed-text, extreu i compara semànticament assignatures d'universitats diferents. El frontend, desenvolupat amb React, ofereix una interfície per visualitzar el percentatge de similitud i una anàlisi qualitativa dels resultats. Aquests demostren que l'eina és capaç d'identificar equivalències amb un alt grau de precisió. El treball estableix les bases per a una eina que pot millorar significativament l'eficiència i transparència del procés Erasmus.

Resumen.

Este proyecto nace para abordar la problemática recurrente y manual de la convalidación de asignaturas en estancias Erasmus, un proceso que genera incertidumbre en los estudiantes y una carga administrativa para los coordinadores. La solución propuesta es un proyecto de un comparador automatizado que, mediante inteligencia artificial, analiza y compara los contenidos de las guías docentes. Este programa se basa en un backend con FastAPI y Python que, usando los modelos Llama3 y nomic-embed-text, extrae y compara semánticamente asignaturas de diferentes universidades. El frontend, desarrollado con React, ofrece una interfaz para visualizar el porcentaje de similitud y un análisis cualitativo de los resultados. Estos demuestran que la herramienta es capaz de identificar equivalencias con un alto grado de precisión. El trabajo establece las bases para una herramienta que puede mejorar significativamente la eficiencia y transparencia del proceso Erasmus.

Abstract.

This project was created to address the recurrent and manual problem of course recognition for Erasmus exchanges, a process that creates uncertainty for students and an administrative burden for coordinators. The proposed solution is an automated comparator project that uses artificial intelligence to analyze and compare the contents of course guides. This program is based on a backend with FastAPI and Python that, using the Llama3 and nomic-embed-text models, semantically extracts and compares subjects from different universities. The frontend, developed with React, provides an interface to view the similarity percentage and a qualitative analysis of the results. These demonstrate that the tool can identify equivalences with a high degree of accuracy. The work lays the foundation for a tool that can significantly improve the efficiency and transparency of the Erasmus process.

1. Índex

2. Índex de taules

3. Índex de figures

4. Introducció

4.1 Descripció del projecte

Quan s'escull una destinació d'Erasmus, en primera instància, no se sap on se'ns permetrà convalidar les assignatures que requerim. És més, s'acaba escollint una destinació sense saber si es podrà convalidar el que es necessita, amb la possibilitat que surti malament o que no s'acabi de convalidar el que es volia, la qual cosa causant incertesa, pèrdues de temps i, en molts casos, resultats insatisfactoris.

El projecte neix per abordar una problemàtica recurrent en l'àmbit acadèmic: la dificultat per trobar assignatures amb continguts similars en estades Erasmus. Actualment, tant els estudiants com els coordinadors s'enfronten a un procés manual i subjectiu, on la convalidació d'assignatures depèn de la revisió dels plans docents i de l'avaluació de la similitud entre els continguts, cosa que provoca que el procés sigui llarg i costós i generi una feina innecessària tant a alumnes com a professors.

Aquesta iniciativa vol resoldre els dubtes d'alumnes que vulguin convalidar assignatures a l'estranger i facilitar la feina del coordinador/professorat encarregat de validar assignatures en estades a altres universitats, possibilitant d'aquesta forma una aplicació per resoldre els dubtes abans d'escollir destinació, al mateix temps que es resolen preocupacions i esforços d'alumnes i coordinadors totalment prescindibles.

4.2 Objectius del projecte

4.2.1 Què volem resoldre?

L'objectiu principal d'aquesta iniciativa és desenvolupar una eina basada en intel·ligència artificial que automatitzi i optimitzi aquest procés. Mitjançant models de llenguatge avançats i tècniques de processament de text, l'aplicació permetrà comparar assignatures de diferents universitats, calcular-ne el grau de similitud i facilitar la presa de decisions tant per als estudiants com per als coordinadors, la qual cosa produeix una millora en la efectivitat i la eficiència a l'hora de convalidar assignatures d'Erasmus.

4.2.2 Quins són els objectius?

Per considerar que hem realitzat un programari útil que permet comparar assignatures de dos universitats diferents amb una sèrie de continguts similars i un percentatge de similitud de l'assignatura, el projecte se centra en dos objectius fonamentals:

1. Resoldre les necessitats dels estudiants: oferir una eina que elimini la incertesa associada a la convalidació d'assignatures durant una estada d'Erasmus, cosa que permetent als estudiants prendre decisions informades abans d'escollir destinació.
2. Facilitar la feina de coordinadors i professors: reduir la càrrega administrativa i el temps dedicat a la validació manual d'assignatures, mitjançant un sistema automatitzat que generi resultats fiables i coherents.

Aconseguir satisfer les necessitats dels objectius fonamentals és essencial, però també volem complir objectius necessaris per ampliar l'abast de l'eina, com:

1. Orientació en l'elecció de destinacions: fomentar que els estudiants utilitzin l'aplicació com a criteri per seleccionar o descartar universitats de destinació.
2. Agilitat en les respostes: permetre que els coordinadors puguin resoldre consultes sobre convalidacions de manera ràpida i precisa.
3. Simplificació del procés Erasmus: contribuir a fer que el procés de mobilitat internacional sigui més transparent i eficient, tant a la Universitat Rovira i Virgili (URV) com en altres institucions.

Amb aquest projecte no només es pretén millorar l'experiència dels estudiants i coordinadors, sinó també establir les bases per a futures innovacions en la gestió acadèmica mitjançant la intel·ligència artificial.

5. Descripció general del problema

5.1 Entorn

5.1.1 Problemàtica acadèmica

L'entorn d'aquest projecte es defineix pel procés de convalidació d'assignatures en el marc del programa Erasmus i/o altres programes, especialment a la Universitat Rovira i Virgili (URV) i altres institucions europees.

Anualment, els estudiants d'Erasmus i altres programes han de comparar manualment els plans docents de la seva universitat d'origen i de destinació per determinar l'equivalència d'assignatures. Aquest procés genera problemàtiques i una càrrega de feina innecessària duta a terme per estudiants i coordinadors acadèmics, a causa de:

- Heterogeneïtat dels formats. Les guies docents es presenten en formats diversos com HTML o diferents plataformes web pròpies de cada universitat, amb diferències en l'estructura, l'idioma (ex. anglès, alemany, francès) i terminologia tècnica. Això dificulta l'extracció i comparació automàtica de continguts.
- Subjectivitat. La validació d'equivalències depèn de la interpretació subjectiva dels coordinadors, cosa que pot generar decisions inconsistents entre diferents facultats o institucions, de manera que augmenta la incertesa per als estudiants.
- Càrrega administrativa. La revisió manual de documents, sovint acompanyada de comunicacions per correu electrònic entre estudiants, coordinadors i administració, pot allargar-se diverses setmanes, la qual cosa incrementa la càrrega de feina i retarda els processos acadèmics.

Aquestes ineficiències no només afecten els estudiants, que poden veure's obligats a triar destinacions sense garanties clares de convalidació, sinó també els coordinadors, que han de gestionar un volum important de sol·licituds sense eines específiques. Així mateix, la manca

d'un sistema únic per comparar assignatures dificulta la planificació d'acords per a les mobilitats.

5.1.2 Entorn tecnològic

El projecte s'emmarca en un entorn tecnològic modern, dissenyat per automatitzar i estandarditzar la comparació d'assignatures mitjançant eines d'intel·ligència artificial (IA) i desenvolupament web. Les principals tecnologies utilitzades són:

- Backend: FastAPI (Python)
  - FastAPI és un framework web i asíncron basat en Python.
- Frontend: React + Vite
  - React és una llibreria JavaScript per a interfícies d'usuari reactives. Vite és una eina de construcció ràpida de desenvolupament web.
- Intel·ligència Artificial: Llama3 (via Ollama) + nomic-embed-text
  - Llama3 és un model de llenguatge més recent de Meta AI.
- Emmagatzematge: SQLite
  - SQLite és una eina de gestió de base de dades relacional lleugera i que no requereix un servidor.

5.2 Necessitats

5.2.1 Necessitats dels usuaris

El projecte involucra dos grups de persones fonamentalment. Els estudiants d'Erasmus o estades amb altres programes i els coordinadors d'aquestes estades.

1. Estudiants Erasmus
   - Es necessita una comparació ràpida i fiable, que estalviï temps innecessari buscant assignatures i enviant correus als professors. Addicionalment, volen saber quines assignatures poden convalidar abans de triar destinació per a la seva tranquil·litat i millora en l'elecció d'universitat.
   - Es volen observar resultats interpretables, és a dir, que les respostes objectives del professor no siguin l'únic feedback que es rebi, necessiten saber per què no se'ls convalida certes assignatures i un percentatge de similitud amb una explicació qualitativa clara resoldria aquests problemes.
   - A més, hi hauria d'haver un clar marcador del quadrimestre on es cursa l'assignatura i els crèdits a què equival, per poder verificar la disponibilitat de les assignatures durant la seva estada.

2. Coordinadors acadèmics
   - Els coordinadors acadèmics reben quantitats de feina costosa i innecessària per part de tots els alumnes de la universitat innecessaris i que podrien ser automatitzats. És a dir, volen que el pes de la seva tasca sigui menys tediosa

per a agilitzar la revisió de continguts de plans docents passant de hores a segons.

- Per aconseguir aquesta revisió ràpida volen la mateixa objectivitat que podria tenir un professor a l'hora d'avaluar les similituds d'una assignatura. La intel·ligència artificial hauria de tenir els mateixos criteris que un professor per declarar l'assignatura convalidable o no.
- Així mateix, seria d'ajuda l'accés a informes detallats que justifiquin les decisions de convalidació, per ajudar-los a decidir generant arguments a favor i en contra a més a més del percentatge de similitud.

5.2.2 Necessitats tècniques

Com bé ja sabem, per implementar aquest projecte farem servir tecnologies diferents que es podran complementar entre elles de forma eficient. Per complir aquest repte, necessitem 5 tècniques essencials per al desenvolupament del projecte:

1. Extracció de dades: l'eina ha de processar guies docents HTML i altres formats, extraient de forma automàtica camps clau com nom, crèdits, temari, competències, etc.
   Per assolir això fem ús de BeautifulSoup, per parsejar HTML, mentre que Llama3 analitza textos no estructurats.

2. Comparació semàntica: cal mesurar la similitud entre temaris, tenint en compte la terminologia tècnica usada i els conceptes equivalents (p. ex.: "Derivades" ≈ "Càlcul diferencial").
   Per tant, ha sigut necessari l'ús d'embeddings (nomic-embed-text), amb el qual calculem similituds cosinus a part de generar una anàlisi qualitativa amb Llama3, que posa també de la seva part per generar un percentatge de similitud.

3. Usabilitat: l'usuari o professor introdueix l'URL i el nom de l'assignatura i rep resultats amb percentatges i explicacions de quines semblances i diferències tenen les respectives assignatures.
   Per aconseguir això, usarem React, que gestiona el flux de l'usuari de forma fluïda amb components visuals com barres de similitud i detalls ampliables. Per l'altre part, FastAPI assegura respostes ràpides via endpoints.

4. Rendiment i escalabilitat: el gran repte de la tardança de resposta, degut a les moltes assignatures que hi ha a una guia docent, pot incrementar els temps a processos molt llargs. Per reduir aquest temps usem una tècnica per revisar si les comprovacions ja s'han fet anteriorment, que es mostraran directament a l'usuari en comptes d'executar-se de nou. A més, SQLite emmagatzema dades localment i evita dependències externes. Finalment, FastAPI gestiona concurrència asíncrona.

5.3 Previsions d'ús

5.3.1 Escenaris d'ús principals

Els escenaris d'ús principals reflecteixen les funcionalitats clau de l'eina, dissenyada per facilitar la convalidació d'assignatures en el context d'Erasmus. Aquests escenaris cobreixen les necessitats dels dos grups d'usuaris principals (estudiants i coordinadors), aprofitant les capacitats tècniques del sistema per automatitzar processos i millorar l'eficiència. Els tres escenaris més rellevants on s'usarà l'eina són:

1. Preparació de l'Erasmus: l'estudiant introdueix l'URL de la seva assignatura i la guia docent de la universitat de destinació, o dues assignatures concretes, per verificar equivalències.
   Per exemple, un estudiant de la URV compara Programació amb assignatures d'una universitat alemanya, filtrant la seva guia docent per trobar quina és la més similar.

2. Validació per coordinadors: els coordinadors utilitzen l'eina per verificar equivalències i generar informes per a expedients acadèmics.
   Com a mostra d'això, podríem usar el mateix exemple: el professor valida la informació generada per l'alumne i la pot contrastar de nou. A més, valorarà ell mateix els resultats per acabar de confirmar o desmentir la convalidació.

3. Planificació acadèmica: les oficines de relacions internacionals identifiquen assignatures compatibles per millorar acords Erasmus.

5.3.2 Previsions de volum

Per complir amb els requisits mínims per fer consultes eficaces i ràpides usem FastAPI i tècniques que s'explicaran posteriorment per reduir el cost, amb temps de resposta més curts per comparació.

També usarem SQLite per a la base de dades, que suporta volums petits sense problemes, amb una taula que emmagatzema dades estructurades (JSON per components i anàlisi). A més, l'arquitectura asíncrona de FastAPI i l'optimització de Llama3 via Ollama asseguren escalabilitat.

No està previst usar un servidor per desplegar el projecte. De moment, en cas d'ús del programa, l'instal·larem el programa de forma local, per usar-lo nosaltres mateixos. Igualment, en cas d'èxit del projecte hauríem de tenir clars els següents aspectes:

- En períodes d'Erasmus, pot pujar exponencialment el nombre d'alumnes que usi l'eina, però en èpoques on no hi hagi programes de mobilització el programa s'usarà molt poc.
- Hauríem d'usar un servidor potent per gestionar les peticions i bases de dades de les consultes que es vagin fent.
- El servidor hauria de ser molt escalable, deixant espai en moments on es facin moltes peticions i reduint costos en moments on no n'hi hagi.

PostgreSQL i MySQL serien les eines més adequades per aconseguir robustesa i escalabilitat del servidor.

6. Anàlisi prèvia

6.1 Recerca d'informació

El procés de recerca d'informació per al desenvolupament del treball ha estat clau per identificar les tecnologies més adequades i dissenyar una solució viable que abordi la problemàtica de la convalidació d'assignatures d'Erasmus. En aquest apartat, es detalla l'anàlisi inicial, les alternatives avaluades i les decisions preses, amb un enfocament en com s'ha escollit la selecció final de les tecnologies usades. La recerca s'ha centrat en equilibrar funcionalitat, costos, privadesa i adequació a un context acadèmic amb recursos limitats.

6.1.1 Recerca d'eines

La recerca d'eines amb intel·ligència artificial se centra a identificar eines i models capaços d'extreure informació estructurada de guies docents (noms, crèdits, temari, competències) i realitzar comparacions semàntiques entre assignatures. L'objectiu era trobar una solució que fos potent, econòmica i compatible amb un entorn local per garantir la privadesa de les dades acadèmiques. Les principals opcions avaluades inclouen:

1. AWS Bedrock: plataforma d'Amazon que ofereix accés a models d'IA com Claude (Anthropic), Titan (AWS) i Llama (Meta) per a tasques de processament de llenguatge natural (NLP), com l'extracció i anàlisi de continguts web.

    - Avantatges: alta escalabilitat, suport per a múltiples models i integració amb l'ecosistema AWS.
    - Inconvenients: costos elevats per a un TFG, especialment per a processament intensiu de guies docents.

2. Amazon SageMaker: eina d'AWS per entrenar i desplegar models d'IA personalitzats, ideal per a tasques d'extracció de dades i anàlisi semàntica.

    - Avantatges: permet crear models adaptats per extreure components específics de guies docents.
    - Inconvenients: requereix coneixements avançats de machine learning i un temps d'entrenament significatiu, es prefereix completar les funcionalitats necessàries de forma eficaç que no invertir la majoria de temps entrenant la intel·ligència artificial.

3. Open AI: plataforma d'OpenAI que ofereix accés a models com GPT-4 i GPT-3.5 per a tasques de processament de llenguatge natural.

    - Avantatges: alta precisió en tasques d'extracció i anàlisi semàntica. Fàcil d'integrar amb LangChain (ChatOpenAI) i amb una API ben documentada.
    - Inconvenients: costos recurrents que podien escalar amb un ús intensiu.

4. Llama3 + Ollama: Llama3 és un model de llenguatge de codi obert desenvolupat per Meta AI, executat localment amb Ollama. Nomic-embed-text proporciona embeddings per a comparacions semàntiques.

    - Avantatges: execució local (ollama pull llama3) que garanteix privadesa i elimina costos recurrents. Llama3 ofereix un rendiment comparable a GPT-3.5 en tasques lingüístiques.
    - Inconvenients: menys precisió que GPT-4 en casos ambigus i temps de resposta més lents en màquines amb recursos limitats. Requereix configuració inicial d'Ollama i dependències.

5. Altres proveïdors:

Proveïdor	**Servei d'IA**	**Ús principal**
Azure	Azure OpenAI Service	Resumir i analitzar contingut web
Google Cloud	Vertex AI	Processar informació estructurada
LangChain	Framework d'agents intel·ligents	Crear agents per navegar i extreure dades

Taula 1. Comparació dels diferents proveïdors.

Després de la realització d'aquesta cerca d'informació, degut als costos nuls i el seu bon rendiment en local, escollim Llama3, en competència amb OpenAI (GPT-4). A més, Llama3, a part de comptar amb millors resultats en proves amb benchmarks, és molt eficient en tasques lingüístiques, com en aquest cas el nostre projecte.

Benchmark	LLama-3 70B	ChatGPT-3.5
Undergraduate level knowledge MMLU (5-shot)	**82.0**	70.0
Graduate level reasoning GPQA (0-shot)	**39,5**	28.1
Code HumanEval (0-shot)	**81.7**	48.1
Grade school math GSM-8K (8-shot, CoT)	**93.0**	57.1
Math problem-solving MATH (4-shot, CoT)	**50.4**	34.1

Figura 1. Comparació Llama3 vs ChatGPT-3.5. [1]

Conjuntament amb Llama3, usarem LangChain, un framework de codi obert que facilita la integració de grans models lingüístics amb aplicacions LLM que ens facilitarà dur a terme el projecte. No he descobert gaires competidors de LangChain, un dels més importants és Semantic Kernel, però he escollit LangChain degut a la seva compatibilitat més alta amb Llama3, és a dir, la decisió ve presa per l'elecció d'Ollama com a agent d'intel·ligència artificial escollit més que per les seves capacitats com a frameworks.

6.1.2 Recerca del backend

La recerca de tecnologies per al backend es va centrar a identificar un framework capaç de gestionar APIs RESTful amb alta eficiència, integrar-se amb eines d'IA (Llama3, nomic-embed-text) i suportar concurrència per a múltiples usuaris. L'objectiu era assegurar temps de resposta baixos i compatibilitat amb l'ecosistema de Python, dominant en intel·ligència artificial. Les alternatives trobades són:

1. FastAPI (Python): Framework web asíncron basat en Python, dissenyat per a APIs RESTful d'alt rendiment, amb suport per Starlette i validació de dades amb Pydantic.
    - Avantatges: rendiment comparable a Node.js o Go gràcies a async/await, ideal per gestionar sol·licituds concurrents. Així mateix, comptem amb integració natural amb l'ecosistema de Python, la qual cosa facilita l'ús de llibreries com LangChain, BeautifulSoup i nomic-embed-text per a tasques d'intel·ligència artificial.
    - Inconvenients: corba d'aprenentatge inicial per a la programació asíncrona. Menor suport per a aplicacions monolítiques en comparació amb Django.

[1] Llama3 vs ChatGPT-3.5, recurs d'internet, link a la webgrafia.

2. Django (Python): Framework web complet de Python, orientat a aplicacions robustes i monolítiques, amb suport per ORM i administració integrada.

    - Avantatges: ecosistema madur i fàcil configuració per a projectes amb bases de dades relacionals. Suporta integració amb llibreries de Python com LangChain.
    - Inconvenients: menys optimitzat per a operacions asíncrones, cosa que augmenta els temps de resposta en comparacions concurrents. Més pesat que FastAPI, amb funcions innecessàries per a una API lleugera.

3. Flask (Python): Framework web lleuger de Python, flexible per a APIs senzilles.

    - Avantatges: simplicitat i baixa corba d'aprenentatge. Compatible amb l'ecosistema de Python.
    - Inconvenients: manca de suport asíncron natiu i validació automàtica de dades, per la qual cosa requereix més codi per gestionar esquemes complexos. Menys escalable per a concurrència alta.

4. Node.js (Express): Framework JavaScript per a APIs ràpides i asíncrones, basat en el motor V8.

    - Avantatges: alta velocitat i suport nadiu per a concurrència. Comunitat àmplia i fàcil integració amb frontends com React.
    - Inconvenients: menor compatibilitat amb l'ecosistema de Python, dominant en IA (ex. LangChain, Llama3). Requereix llibreries addicionals per a tasques d'IA, cosa que augmenta la complexitat d'integració.

5. Spring Boot (Java): Framework Java per a aplicacions empresarials robustes, amb suport per APIs RESTful i bases de dades relacionals.

    - Avantatges: alta escalabilitat i suport per a entorns complexos. Ecosistema madur per a projectes a gran escala.
    - Inconvenients: corba d'aprenentatge pronunciada i configuració complexa. Menor compatibilitat amb llibreries d'intel·ligència artificial de Python, per la qual cosa requereix ponts d'integració (ex. gRPC). Temps de desenvolupament elevat per a un TFG.

6.1.3 Recerca del frontend

La recerca de tecnologies per al frontend es va centrar en trobar una eina que permetés crear una interfície d'usuari intuïtiva, reactiva i accessible per a usuaris no tècnics (estudiants i coordinadors). L'objectiu era proporcionar una experiència fluida per introduir URLs, noms

d'assignatures i visualitzar resultats detallats (puntuacions, explicacions, historial). Les alternatives avaluades són:

1. React + Vite: React és una llibreria JavaScript per a interfícies d'usuari modulars i reactives, mentre que Vite és una eina de construcció que optimitza el desenvolupament amb compilacions ràpides i un servidor de desenvolupament eficient.
    - Avantatges: React permet crear components reutilitzables i gestionar estats dinàmics per mostrar resultats en temps real, com barres de similitud i taules de resultats. Per l'altra banda, Vite accelera el desenvolupament amb temps de compilació mínims i suport per a mòduls ES, ideal per a un TFG amb terminis ajustats. Finalment, comptem amb una comunitat àmplia i llibreries com Axios per a peticions HTTP, la qual cosa facilita la integració amb el backend FastAPI.
    - Inconvenients: corba d'aprenentatge inicial per a la gestió d'estats i components.

2. Vue.js: Framework JavaScript lleuger per a interfícies reactives, similar a React, però amb una sintaxi més senzilla.
    - Avantatges: fàcil d'aprendre i configurar, amb una comunitat creixent. Suporta eines com Vite per a compilacions ràpides.
    - Inconvenients: menor ecosistema de llibreries en comparació amb React, cosa que podria limitar la integració de funcions complexes com la visualització de gràfics o l'exportació de dades. Menor suport comunitari per a projectes acadèmics.

3. Angular: Framework JavaScript complet desenvolupat per Google, orientat a aplicacions empresarials complexes.
    - Avantatges: estructura robusta i suport per a aplicacions a gran escala. Inclou eines integrades per a gestió d'estats i formularis.
    - Inconvenients: corba d'aprenentatge pronunciada i configuració complexa, inadequada per a un TFG amb temps limitat. Genera codi més pesat, la qual cosa augmenta els temps de càrrega per als usuaris.

6.2 Conclusions tecnològiques

Les tecnologies seleccionades per al projecte (FastAPI, React + Vite, Llama3, nomic-embed-text, SQLite, LangChain, BeautifulSoup) reflecteixen un equilibri entre funcionalitat, costos, privadesa i adequació en un context acadèmic amb recursos limitats. A continuació, s'explica breument per què s'han escollit aquestes eines, evitant redundàncies amb l'anàlisi prèvia i centrant-se en els avantatges clau que les fan òptimes per al TFG.

6.2.1 Backend: FastAPI (Python)

Hem escollit FastAPI per la seva alta eficiència en la gestió d'APIs RESTful, gràcies al suport asíncron (async/await) que garanteix temps de resposta baixos per a múltiples usuaris. La seva integració amb l'ecosistema de Python permet una compatibilitat amb eines d'intel·ligència artificial com LangChain i nomic-embed-text, essent més lleuger i ràpid que Django o Flask, i més alineat amb Python que Node.js o Spring Boot. La validació de dades amb Pydantic i la documentació automàtica acceleren el desenvolupament, ideals per a un TFG.

6.2.2 Frontend: React + Vite

La selecció de React + Vite està basada en la seva capacitat per crear una interfície intuïtiva i reactiva, la qual cosa permet als usuaris (estudiants i coordinadors) introduir URLs i visualitzar resultats amb components dinàmics. React ofereix modularitat i suport comunitari ampli, superant Vue.js en ecosistema i Angular en simplicitat. Vite accelera el desenvolupament amb compilacions ràpides, reduint els temps de càrrega i facilitant la integració amb FastAPI via Axios. Això assegura una experiència d'usuari fluida i accessible.

6.2.3 Intel·ligència Artificial: Llama3 i nomic-embed-text

Llama3 ha sigut usat pel el codi obert i execució local via Ollama, garantint privadesa i eliminant costos recurrents, a diferència d'OpenAI o AWS Bedrock. Ofereix un rendiment comparable a GPT-3.5 per a l'extracció de components estructurats i anàlisis qualitatives, segons benchmarks. Nomic-embed-text, lleuger i eficient, calcula similituds cosinus en texts multilingües, i supera models més complexos com BERT en simplicitat i adequació a entorns locals.

6.2.4 Emmagatzematge: SQLite

SQLite s'ha seleccionat per la seva simplicitat i portabilitat, ideal per a volums moderats de dades sense necessitat de servidors complexos, a diferència de PostgreSQL o MySQL. La seva integració amb FastAPI permet emmagatzemar l'historial de comparacions amb traçabilitat, cobrint els requisits mínims del projecte amb un manteniment ínfim.

6.2.5 Altres eines

- LangChain: escollit per la seva compatibilitat directa amb Llama3 i facilitat per estructurar fluxos de treball amb prompts i agents. Supera Semantic Kernel en integració amb Python, comunitat activa i simplicitat per a tasques com l'extracció i comparació semàntica, essencials per al projecte.

- BeautifulSoup: preferit sobre Scrapy per la seva lleugeresa i facilitat d'ús en el parsing d'HTML, adequat per a guies docents en formats web.

6.3 Anàlisi de tecnologies

Per tant, de forma resumida, aquestes seran les tecnologies usades i les raons per les quals les utilitzem:

Component	Tecnologia	Justificació
LLM per a comprensió de textos	LLaMA3 via Ollama	Alta qualitat, local, sense cost
Embeddings	nomic-embed-text	Optimitzat per textos acadèmics
Framework d'agents	LangChain	Integració amb LLM i eines
Web scraping	BeautifulSoup	Robust per accedir
Comparació de similituds	Sklearn + LLM	Simple i adequat per a comparacions puntuals.
Backend	FastAPI	API simple
Frontend	React	Interfície senzilla i visual
Emmagatzematge	SQLite	Lleuger i simple

Taula 2. Tecnologies usades al codi.

7. Requisits

En aquest apartat es documenten els requisits funcionals i no funcionals del sistema i es detallen les funcionalitats necessàries per automatitzar la comparació d'assignatures Erasmus i les característiques tècniques que asseguren privadesa, usabilitat i eficiència. Els requisits funcionals s'expressen com a funcionalitats prioritzades amb el mètode MoSCoW, i s'inclou una anàlisi de riscos per identificar obstacles i mesures correctives. L'enfocament se centra en una aplicació simple i útil, alineada amb els recursos limitats d'un TFG acadèmic.

7.1 Requisits funcionals

Els requisits funcionals defineixen les funcionalitats que el sistema ha de proporcionar per complir els objectius del projecte: extreure, comparar i analitzar assignatures de guies docents de manera automatitzada, amb una interfície intuïtiva per a estudiants i coordinadors. Les funcionalitats es basen en un agent d'intel·ligència artificial que utilitza LangChain i Llama3 per processar informació, amb suport per a traducció i emmagatzematge de resultats.

Per tant, els requisits funcionals del projecte seran:

- Scraping amb LangChain: extreure contingut estructurat (noms, crèdits, temari, competències) de guies docents a partir d'URLs, utilitzant BeautifulSoup.

- Anàlisi amb Llama3: identificar noms d'assignatures i, opcionalment, crèdits i quadrimestre, mitjançant prompts estructurats (extract_subject_theme, extract_subjects_with_llm).

- Comparació d'assignatures: comparar assignatures seleccionades, generant un percentatge de similitud amb nomic-embed-text (compute_embedding_similarity) i una explicació qualitativa (qualitative_analysis).

- Traducció automàtica: traduir noms i continguts a anglès amb Llama3 per a comparacions coherents en contextos multilingües.

- Buscador de graus i assignatures: permetre la cerca de graus i assignatures disponibles per facilitar la selecció (/subjects_list).

- Emmagatzematge i visualització: guardar resultats de comparacions a SQLite (comparisons.db) i mostrar-los a la interfície React (ComparisonResults), amb opció d'historial (/comparison_history).

- API eficient: proporcionar una API RESTful senzilla amb FastAPI (/compare, /comparison_history) per a una integració fàcil i ràpida.

L'objectiu és una aplicació simple, on les funcionalitats essencials siguin fàcils d'utilitzar, evitant una API complexa o difícil de recordar, com es reflecteix en el disseny de les rutes de FastAPI i la interfície React.

7.2 Requisits no funcionals

Els requisits no funcionals defineixen les qualitats operatives i tècniques del sistema, assegurant que compleixi els estàndards de privadesa, eficiència i usabilitat requerits per un context acadèmic:

- Privacitat: totes les dades (guies docents, comparacions) es processen localment amb Llama3 i Ollama (ollama pull llama3), sense dependència de serveis al núvol, garantint la confidencialitat de les dades acadèmiques.

- Portabilitat: el sistema és executable en qualsevol ordinador amb Python, Ollama i dependències instal·lades (pip install langchain fastapi), amb SQLite com a base de dades lleugera (comparisons.db).

- Eficiència: les comparacions han de retornar resultats en menys de 10 minuts, optimitzades amb les tècniques usades (com comprovació comparació anterior i comparació temàtica) i FAISS per a indexació ràpida de similituds.

- Usabilitat: la interfície React (ComparisonInputs, ComparisonResults) és clara, intuïtiva i accessible per a usuaris no tècnics, amb visuals com barres de similitud (getSimilarityColor) i sense requerir formació prèvia.

- Modularitat: el sistema permet substituir Llama3 o nomic-embed-text per altres models o embeddings, gràcies a l'estructura flexible de LangChain (OllamaEmbeddings).

- Mantenibilitat: el codi està estructurat en mòduls (ex. extract_urv_contents, compute_embedding_similarity) i documentat per facilitar futures actualitzacions.

7.3 MoSCoW method

El mètode MoSCoW és una tècnica de gestió de priorització utilitzada en desenvolupament de programari que classifica les funcionalitats en quatre categories segons la seva importància: Must Have (imprescindibles), Should Have (desitjables), Could Have (opcionals) i Won't Have (excloses en aquesta fase). Per tant, la taula MoSCoW per a avaluar les funcionalitats importants queda així:

Figura 2. MoSCoW. [2]

Si es compleixen aquests requisits significa que el programari ha assolit la idea del projecte satisfactòriament.

[2] MoSCoW template, recurs d'internet, link a la webgrafia.

8. Anàlisi dels requisits funcionals

Aquest apartat analitza els requisits funcionals del sistema, detallant les interaccions entre els usuaris i el sistema mitjançant el diagrama de casos d'ús, l'estructura de les entitats amb el diagrama de classes i el flux d'operacions amb els diagrames de seqüència. S'integra amb les taules d'anàlisi de requeriments, tecnologies i riscos proporcionades, de manera que assegura una visió completa i coherent amb els objectius del projecte: automatitzar la comparació d'assignatures Erasmus amb una aplicació senzilla, privada i eficient.

8.1 Diagrama de classes

El diagrama de classes següent defineix l'estructura de dades i serveis del sistema i reflecteix com s'organitzen i interactuen els diferents components de l'aplicació per automatitzar la comparació d'assignatures Erasmus:

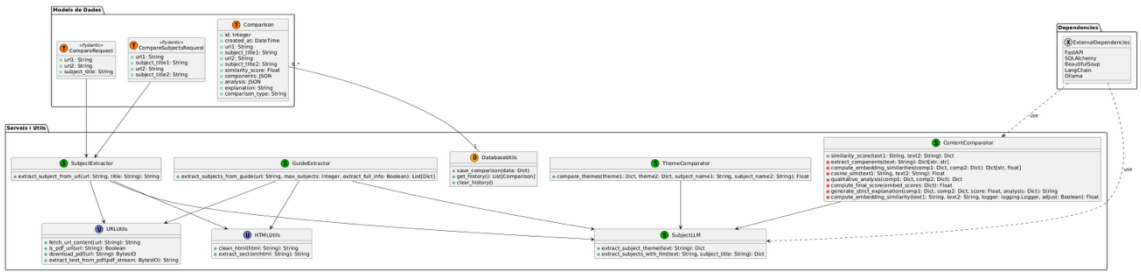


Figura 3. Diagrama de classes. [3]

S'ha dividit en tres paquets principals per afavorir la modularitat, la claredat i el manteniment:

8.1.1 Models de Dades

Aquest paquet conté les entitats que defineixen les estructures d'entrada i sortida de l'API, així com el model principal per emmagatzemar comparacions:

- Comparison: representa una comparació entre dues assignatures. Inclou informació com les URL d'origen, els títols, la puntuació de similitud, l'anàlisi qualitativa i els components comparats (objectius, continguts, competències...).
- CompareRequest i CompareSubjectsRequest: models de dades usats com a entrada a l'API, construïts amb Pydantic per validar i parsejar les dades d'entrada (títols i URLs de les assignatures).

[3] PlantUML, recurs d'internet, link a la webgrafia.

8.1.2 Serveis i Útils

Conté els serveis funcionals del sistema:

- SubjectExtractor: s'encarrega d'extreure informació d'una guia docent. Fa ús d'utilitats HTML i pot processar contingut amb LLMs.
- GuideExtractor: permet extreure múltiples assignatures d'una mateixa guia docent, amb opció de recollir informació detallada.
- SubjectLLM: interacciona amb Llama3 a través de LangChain per reconèixer assignatures dins del text i extreure'n els components semàntics.
- ContentComparator: compara dues assignatures a partir del contingut extret. Combina embeddings (nomic-embed-text), càlculs de similitud (cosinus) i una anàlisi qualitativa. També genera una explicació textual sobre la comparació.
- ThemeComparator: ofereix una comparació basada en temàtiques detectades amb LLM.
- URLUtils i HTMLUtils: contenen funcionalitats de baix nivell per descarregar i processar contingut des de la web.
- DatabaseUtils: gestiona la persistència de les comparacions a la base de dades SQLite, així com l'accés a l'historial i la seva esborrada.

8.1.3 Dependències

Aquest paquet identifica les dependències externes utilitzades per implementar el sistema:

- FastAPI: per crear l'API RESTful.
- SQLAlchemy: per gestionar la persistència a SQLite.
- BeautifulSoup: per fer web scraping.
- LangChain i Ollama: per facilitar la interacció amb Llama3 i la gestió de prompts.
- Altres com sklearn es considera també implicat, encara que no està representat explícitament en el diagrama.

8.1.4 Relacions

El diagrama també mostra les relacions funcionals entre les classes:

- Els models CompareRequest i CompareSubjectsRequest utilitzen SubjectExtractor per obtenir el contingut a comparar.
- SubjectExtractor depèn de les utilitats de URLUtils, HTMLUtils i els serveis de LLM.
- ContentComparator i ThemeComparator fan ús del model semàntic proporcionat per SubjectLLM.
- DatabaseUtils s'encarrega d'emmagatzemar instàncies de Comparison.

Aquest disseny modular i orientat a serveis facilita l'escalabilitat i permet una fàcil substitució de components (per exemple, canviar el model d'embeddings o el model LLM) sense alterar el funcionament general de l'aplicació. També assegura una separació clara entre les capes de dades, lògica i persistència, reforçant el manteniment i la claror arquitectònica del projecte.

8.2 Diagrama de seqüència

El diagrama de seqüència descrit representa els fluxos d'interacció entre els diferents components del sistema per a l'automatització de la comparació d'assignatures Erasmus. Aquest diagrama, elaborat amb PlantUML, detalla quatre casos d'ús principals: comparació d'assignatures individuals, comparació d'una assignatura amb una guia docent, consulta de l'historial de comparacions i esborrat de l'historial. A continuació, s'explica en detall cada cas d'ús, descrivint les interaccions entre l'usuari, el frontent, el backend, la base de dades, el servidor web extern, el model LLM i el model d'embeddings:

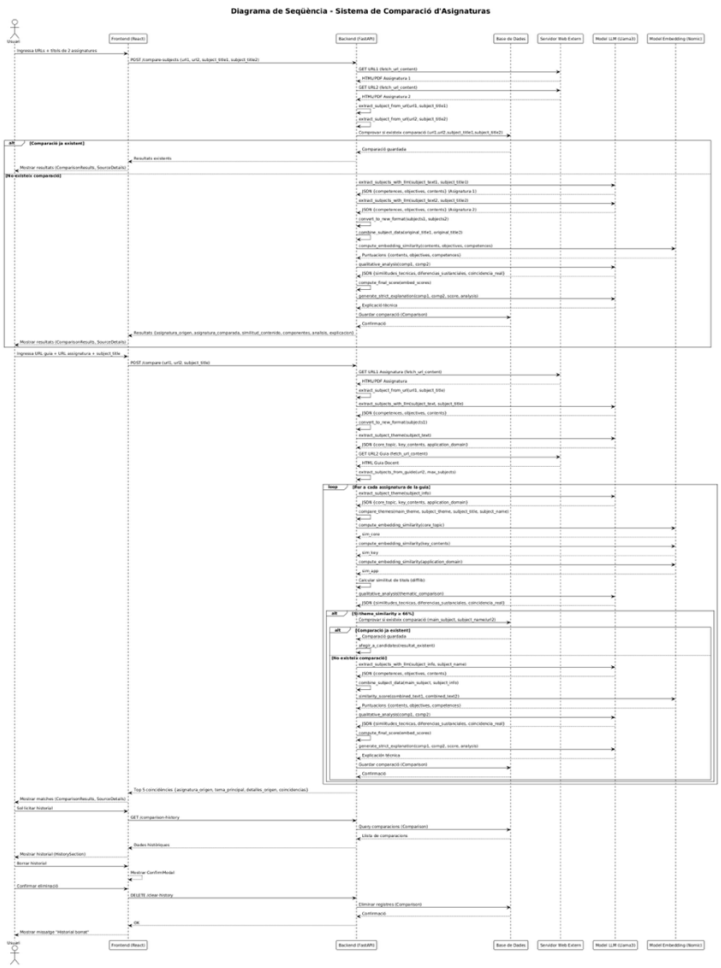


Figura 4. Diagrama de seqüència. [4]

[4] PlantUML, recurs d'internet, link a la webgrafia.

8.2.1 Comparació d'assignatures individuals

Aquest cas d'ús permet a l'usuari comparar dues assignatures específiques introduint les seves URLs i títols.

- L'usuari introdueix al Frontend (component ComparisonInputs) les URLs (url1, url2) i els títols (subject_title1, subject_title2) de dues assignatures.
- El Frontend envia una petició POST a l'endpoint /compare-subjects del Backend amb les dades proporcionades.
- El Backend utilitza la funció fetch_url_content per descarregar el contingut de les URLs des del Servidor Web Extern.
- Per a cada assignatura, el Backend executa extract_subject_from_url per processar el contingut i extreure'n informació rellevant.
- Abans de comparar i obtenir un JSON, es comprova a la base de dades a veure si aquesta comparació ja s'ha fet prèviament. En cas que ja s'hagi dut a terme, es mostrarà; en cas contrari, seguirà el seu procés habitual.
- A continuació, el Backend invoca el Model LLM (Llama3) amb la funció extract_subjects_with_llm per obtenir un JSON estructurat amb els components de l'assignatura (competències, objectius i continguts).
- Els resultats es converteixen al format adequat amb convert_to_new_format i es combinen amb combine_subject_data per preparar la comparació.
- El Backend utilitza el Model Embedding (Nomic) amb compute_embedding_similarity per calcular la similitud entre els continguts, objectius i competències de les dues assignatures.
- Es realitza una anàlisi qualitativa mitjançant el Model LLM amb qualitative_analysis, que retorna un JSON amb similituds tècniques, diferències substancials i una puntuació de coincidència real.
- El Backend calcula una puntuació final amb compute_final_score i genera una explicació tècnica amb generate_strict_explanation a través del Model LLM.
- La comparació es desa a la Base de Dades (model Comparison) i es retorna al Frontend amb els resultats, incloent-hi la similitud, components, anàlisi i explicació.
- El Frontend mostra els resultats a l'usuari mitjançant els components ComparisonResults i SourceDetails.

Aquest flux garanteix una comparació detallada i tècnica de dues assignatures de dues carreres diferents, comparant el seu contingut, metodologia i competències i generant una explicació i percentatge de similitud on participen l'LLM i el sklearn.

8.2.2 Comparació amb una guia docent

Aquest cas d'ús permet comparar una assignatura específica amb múltiples assignatures extretes d'una guia docent.

- L'usuari introdueix al Frontend l'URL d'una assignatura (url1), l'URL d'una guia docent (url2) i el títol de l'assignatura (subject_title).
- El Frontend envia una petició POST a l'endpoint /compare del Backend.
- El Backend descarrega el contingut de l'assignatura (url1) amb fetch_url_content i el processa amb extract_subject_from_url.
- El Model LLM extreu els components estructurats (competències, objectius, continguts) amb extract_subjects_with_llm i la temàtica principal amb extract_subject_theme (temes clau, continguts i domini d'aplicació).
- El Backend descarrega el contingut de la guia docent (url2) i utilitza extract_subjects_from_guide per identificar fins a 10 assignatures.
- Per a cada assignatura de la guia:
    - El Model LLM extreu la temàtica amb extract_subject_theme.
    - El Backend compara la temàtica principal amb la de l'assignatura amb compare_themes, utilitzant el Model Embedding per calcular similituds en core_topic, key_contents i application_domain, i difflib per a la similitud dels títols.
    - Si la similitud temàtica és ≥ 66%, es fa una anàlisi detallada:
        - Comprova al backend si ja s'han comparat aquestes assignatures i si és el cas les mostra.
        - S'extreuen els components complets amb extract_subjects_with_llm.
        - Es combinen les dades amb combine_subject_data.
        - El Model Embedding calcula la similitud amb similarity_score.
        - El Model LLM realitza una anàlisi qualitativa amb qualitative_analysis i genera una explicació amb generate_strict_explanation.
        - La comparació es desa a la Base de Dades (model Comparison).
- El Backend retorna al Frontend les 5 millors coincidències, incloent-hi l'assignatura d'origen, la temàtica principal, els detalls i les coincidències.
- El Frontend mostra els resultats a l'usuari mitjançant ComparisonResults i SourceDetails.

Aquest flux permet identificar assignatures equivalents dins d'una guia docent, filtrant primer per similitud temàtica i després realitzant una comparació detallada per a les assignatures més rellevants. A més, en cas que aquesta comparació ja s'hagi realitzat, es mostrarà sense executar el codi de nou. Posteriorment, generaria una explicació i un percentatge que reflecteixi la similitud de la comparació.

8.2.3 Consulta de l'historial de comparacions

Aquest cas d'ús permet a l'usuari veure l'historial de comparacions prèvies.

- L'usuari sol·licita l'historial al Frontend (component HistorySection).
- El Frontend envia una petició GET a l'endpoint /comparison-history del Backend.

- El Backend consulta la Base de Dades per obtenir totes les comparacions (model Comparison).
- La Base de Dades retorna la llista de comparacions.
- El Backend envia les dades al Frontend, que les mostra a l'usuari mitjançant HistorySection, incloent-hi detalls com les URLs, títols, puntuacions i explicacions.

Proporciona a l'usuari un accés ràpid i visual a les comparacions anteriors, millorant la traçabilitat i l'experiència d'usuari.

8.2.4 Esborrat de l'historial

Aquest cas d'ús permet a l'usuari eliminar l'historial de comparacions.

- L'usuari selecciona l'opció d'esborrar l'historial al Frontend (HistorySection).
- El Frontend mostra un ConfirmModal per confirmar l'acció.
- L'usuari confirma l'eliminació.
- El Frontend envia una petició DELETE a l'endpoint /clear-history del Backend.
- El Backend elimina tots els registres de la taula Comparison de la Base de Dades.
- La Base de Dades confirma l'eliminació.
- El Backend retorna un missatge d'èxit al Frontend.
- El Frontend mostra a l'usuari un missatge de confirmació ("Historial borrado").

Permet a l'usuari gestionar l'historial de manera senzilla, mantenint el control sobre les dades emmagatzemades.

8.3 Diagrama de casos d'ús

El diagrama de casos d'ús, generat amb PlantUML, és una representació gràfica clau en l'anàlisi dels requisits funcionals. Aquest diagrama mostra les interaccions entre els actors (usuaris o sistemes externs) i les funcionalitats que el sistema proporciona, ajudant a identificar i organitzar els requisits funcionals:

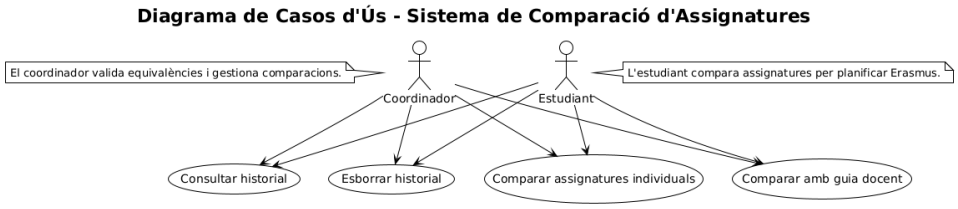


Figura 5. Diagrama de casos d'ús. [5]

El diagrama, tal com es mostra al codi PlantUML, inclou els següents elements:

[5] PlantUML diagrams, recurs d'internet, link a la webgrafia.

- Actors:
  - Estudiant: representa l'estudiant que utilitza el sistema per comparar assignatures i planificar la seva estada Erasmus. La seva nota associada reflecteix el seu rol principal, alineat amb les necessitats descrites a "Necessitats dels usuaris", on mencionem que els estudiants necessiten una comparació ràpida i fiable per triar destinació i reduir incertesa.
  - Coordinador: representa el coordinador acadèmic, que utilitza el sistema per validar equivalències d'assignatures i gestionar comparacions. La nota associada menciona la necessitat de reduir la càrrega administrativa i agilitzar revisions, com es detalla a "Necessitats dels usuaris".
- Casos d'ús:
  - Comparar assignatures individuals: permet a l'usuari comparar dues assignatures específiques introduint les seves URLs i títols. S'extreuen components amb Llama3 i calcula similituds amb nomic-embed-text.
  - Comparar amb guia docent: permet comparar una assignatura amb una guia docent completa, que pot contenir múltiples assignatures. Comparem totes les assignatures de la guia docent, però no sense abans filtrar per similitud temàtica (≥66%) per optimitzar temps de resposta.
  - Consultar historial: permet visualitzar l'historial de comparacions prèvies, on es consulta la base de dades SQLite amb /comparison-history.
  - Esborrar historial: permet eliminar l'historial de comparacions, amb una confirmació via ConfirmModal i una petició DELETE a /clear-history.

8.4 Anàlisi de requeriments

Taula de tasques per implementar els requisits funcionals:

Tasca	Temps	Prioritat	Observacions
Instal·lació Ollama i models	2h	Alta	Instal·lació local i proves bàsiques
Extracció del contingut de guies	30h	Alta	BeautifulSoup
Cerca automatitzada de guies docents	30h	Alta	BeautifulSoup
Embeddings i comparació de continguts	35h	Alta	nomic-embed-text + sklearn
Disseny i implementació interfície	10h	Mitjana	React + Vite
Historial i comparacions	5h	Baixa	Opcional, SQLite

Testing i millores	10h	Alt	Optimització i validació resultats

Taula 3. Temps de tasques.

8.5 Anàlisi de riscos (amb matriu de riscos)

Podem tenir diversos factors de risc:

1. El model no entén bé els continguts acadèmics.
2. Errades en l'scraping de pàgines universitàries.
3. Problemes de rendiment amb arxius grans.
4. Mala formació de les guies docents (formats inhabituals).
5. Dificultat d'ús.
6. Manca de documentació tècnica.
7. Retards en la realització del projecte degut a la seva complexitat.

Per tant, diversos factors poden afectar el projecte:

Risc Escala: Molt baix – Baix – Moderat – Alt – Molt alt	Probabilitat (1-5)	Impacte (1-5)	Puntuació (1-25)
1. Model no fiable	Moderat	Molt alt	15
2. Errors scraping	Molt alt	Moderat	12
3. Arxius grans	Moderat	Molt alt	15
4. Guies docents mal formades	Moderat	Alt	12
5. Dificultat d'ús	Baix	Molt alt	10
6. Manca documentació	Moderat	Moderat	9
7. Retards per complexitat	Moderat	Moderat	9

Taula 4. Matriu de riscos.

Podem intentar corregir el màxim possible els riscos aplicant mesures correctives:

1. Canviar embeddings o afinar prompts.
2. Afegir suport a més dominis manualment.
3. Preprocessar textos i dividir per seccions.
4. Extracció de les guies docents més treballada.
5. Enfocar-se en simplicitat i claredat de la interfície.
6. Preveure temps per documentar i provar.
7. Gestió de temps i limitar funcionalitats opcionals.

9. Disseny

9.1 Decisions de disseny del codi

Hem de prendre moltes decisions a l'hora de realitzar el programa.

Per començar, hem pres decisions de disseny sobre com ha de funcionar l'aplicació:

- Crec que la forma de fer més fàcil la comparació d'assignatures seria agafar una assignatura d'un pla d'estudis d'un grau, extreure el seu contingut, agafar l'altre pla d'estudis i extreure els continguts i comparar l'assignatura que volem convalidar amb totes les assignatures d'un altre grau, trobant així quines són les assignatures més similars. Per tant, farem una comparació d'una assignatura del nostre grau amb moltes assignatures d'un altre grau.
- És molt important mantenir un criteri fiable de la similitud de l'assignatura, assegurant que es compleix de forma satisfactòria la convalidació tant per part del professor com per part de l'alumne. El mètode que creiem més adient és aquell que contempla que, com més temari trobem similar o frases similars sobre el contingut que hi hagi entre una assignatura i una altre més alt serà el percentatge.
- Amb una base de dades milloraríem els temps d'optimització i no es repetirien comparacions ja fetes. Preguntar a la base de dades si aquesta operació ja s'ha fet i mostrar-la directament ajudaria al codi a ser més ràpid i eficient.

9.2 Decisions de disseny de l'aplicatiu web

A més, també es important remarcar les decisions de disseny de l'aplicació:

- Volem aconseguir una interfície clara, neta, fàcil d'utilitzar i intuïtiva. Per això mateix, tan bon punt s'obri l'aplicació trobarem un cercador de guies docents, on buscarem la guia docent que necessitem. A més, també trobarem altres opcions com l'historial.
- Posteriorment a seleccionar la guia docent, s'extrauran totes les assignatures i es prepararan per comparar-les.
- Després, seleccionarem una assignatura de la guia docent (la que es vulgui convalidar) i es compararà l'assignatura amb una altra guia docent que seleccionarem afegint l'URL.
- Més tard, es mostrarà l'assignatura seleccionada, amb el seu nom i el contingut, i a sota de la pantalla tindrem les assignatures de l'altre guia docent amb un percentatge al costat que dirà fins a quin punt s'assembla a l'assignatura que volem convalidar.
- Finalment, podrem tornar al menú principal, on trobarem de nou el menú per seleccionar la guia docent que vulguem de nou.

9.3 Arquitectura de l'aplicació

L'aplicació segueix una arquitectura client-servidor estàndard, amb components clars i separats per a frontend, backend i persistència de dades:

- Frontend: desenvolupat amb React, utilitza components com ComparisonInputs, ComparisonResults, SourceDetails, ConfirmModal i HistorySection. Aquests components gestionen la interfície d'usuari, amb hooks com useComparison per gestionar estat i crides a l'API. El disseny és responsiu, amb CSS per a estils moderns i llegibles, utilitzant flex i grid per a la disposició.
- Backend: implementat amb FastAPI, un framework asíncron per a APIs RESTful, amb endpoints com /compare, /compare-subjects, /comparison-history i /clear-history. Integra un LLM per a tasques com l'extracció de temes i anàlisis qualitatives, afegint intel·ligència artificial al procés.
- Base de dades: utilitza SQLite, una base de dades relacional lleugera adequada per a aplicacions web petites o mitjanes. La taula comparisons emmagatzema resultats de comparacions, assegurant persistència i recuperació d'historial.

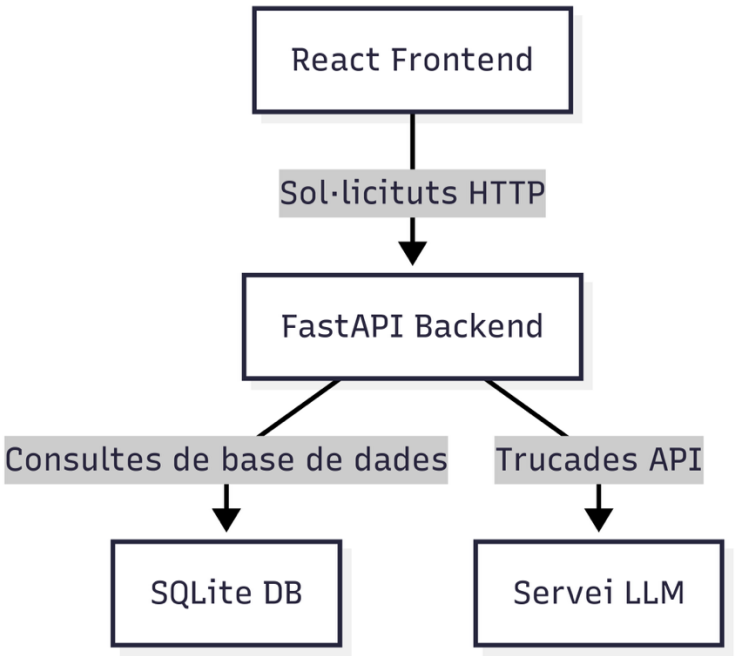


Figura 6. Arquitectura de l'aplicació. [6]

Aquesta arquitectura permet una separació clara de responsabilitats, i facilita el manteniment, l'escalabilitat i la integració de noves funcions, com l'anàlisi avançada amb intel·ligència artificial.

[6] Mermaid.js, recurs d'internet, link a la webgrafia

9.4 Disseny de la persistència de dades

La persistència de dades es fonamenta en una base de dades SQLite, amb un esquema definit mitjançant SQLAlchemy. La taula principal, comparisons, emmagatzema tota la informació relativa a cada comparació realitzada, amb el següent esquema:

Camp	Tipus	Descripció
id	Integer	Identificador únic
created_at	DataTime	Timestamp de creació de la comparació
url1	String	URL de la assginatura origen
subject_title1	String	Títol de l'assignatura origen
guide_url	String	URL de la guia docent
url2	String	URL de la assignatura destinació
subject_title2	String	Títol de l'assignatura de destinació
similarity_score	Float	Puntuació de similitud general
components	JSON	Puntuacions per components
analysis	JSON	Anàlisi detallada del LLM (similituds, diferències)
explanation	String	Explicació en text de la comparació
comparison_type	String	Tipus de comparació (compare o compare-subjects)
detalles_origen	JSON	Detalls de l'assignatura d'origen
detalles_comparada	JSON	Detalls de l'assignatura comparada
theme_similarity	Float	Similitud del tema de l'assignatura

Taula 5. Atributs base de dades.

Aquest esquema permet emmagatzemar informació detallada de cada comparació, facilitant la recuperació per a l'historial. Els endpoints /comparison-history i /clear-history gestionen l'accés i l'esborrat de dades, assegurant una gestió eficient de l'historial.

9.5 Prototipats

Aquests prototips tenen com a objectiu explorar diverses opcions de disseny abans de la implementació final, millorant la usabilitat i l'experiència d'usuari.

He realitzat 3 prototips diferents amb Freeform (una pissarra digital desenvolupada per Apple) enfocats en diverses formes de representar l'aplicació.

- Prototip 1:

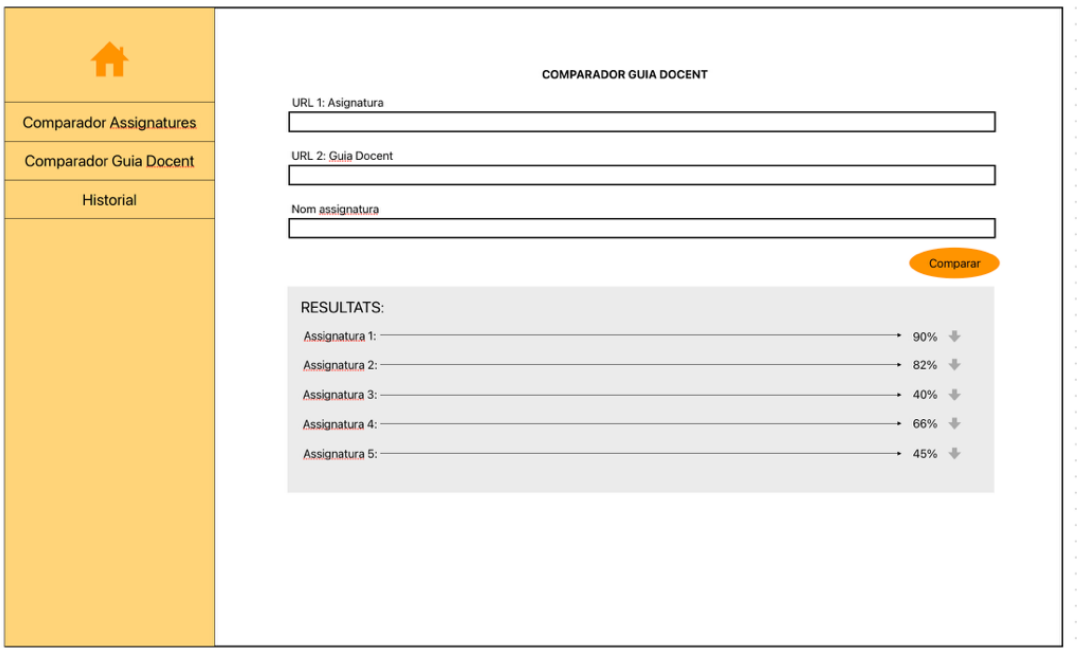


Figura 7. Prototip 1.

- Prototip 2:

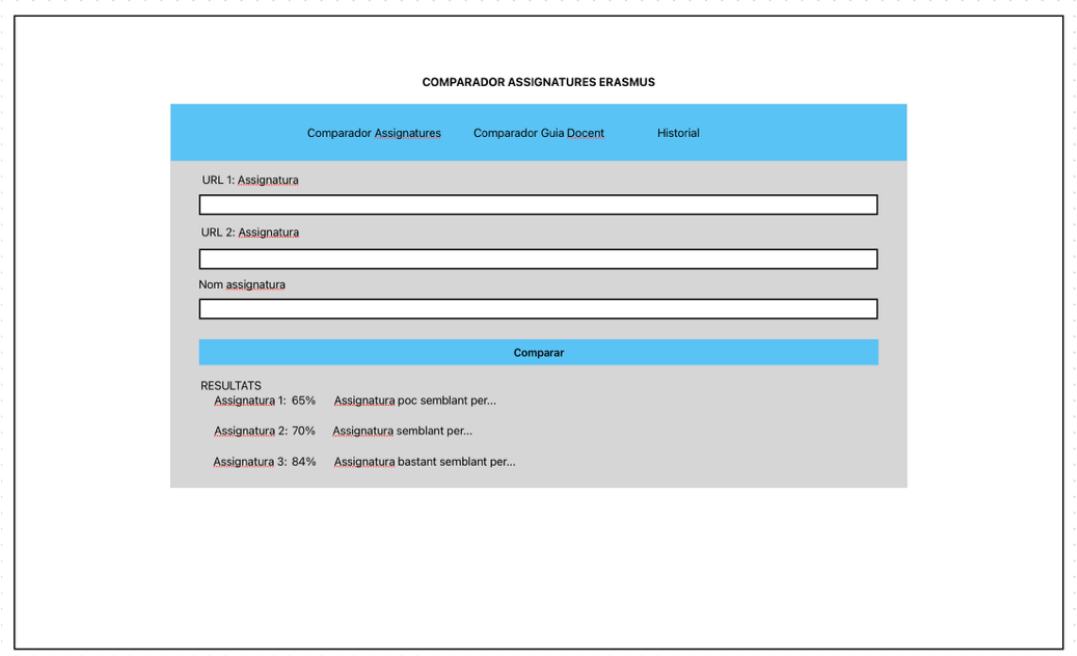


Figura 8. Prototip 2.

- Prototip 3:

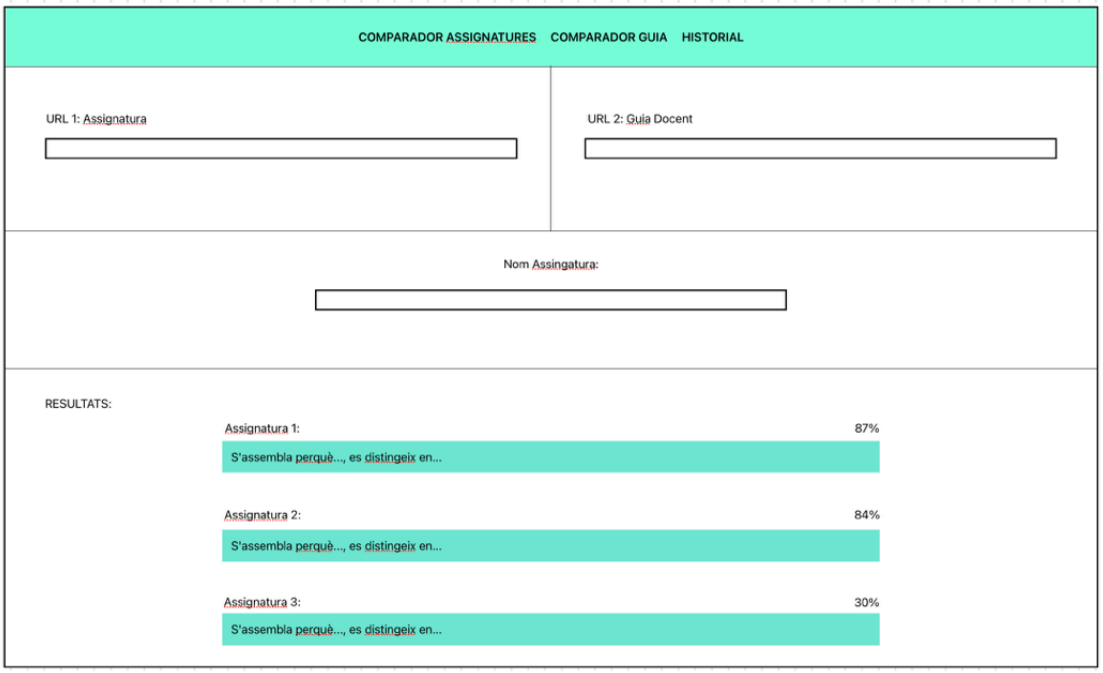


Figura 9. Prototip 3.

Aquests prototips ajudaran a donar idees, identificar millores en el flux d'usuari, a millorar la presentació de resultats i la navegació, i asseguren que l'aplicació final sigui intuïtiva i eficient.

9.6 Disseny de la interfície

Després de realitzar una breu enquesta no oficial a sis coneguts sobre quins prototips els hi semblaven més intuïtius i eficaços a l'hora de comparar les assignatures, he obtingut els següents resultats:

- Prototip 1: 50% (3 persones)
- Prototip 2: 50% (3 persones)
- Prototip 3: 0%

Per escollir el prototip final, he realitzat una barreja dels dos primers prototips mostrats anteriorment, com les enquestes han suggerit.

Per preferència personal, he escollit com a base el prototip 2 usant el color blau. Com bé he comentat, he afegit coses del primer prototip que van agradar, com la fletxa per desplegar els resultats i els subapartats de forma més estructurada i clara, respectant la forma i simplicitat del primer prototip.

El resultat final es el següent:

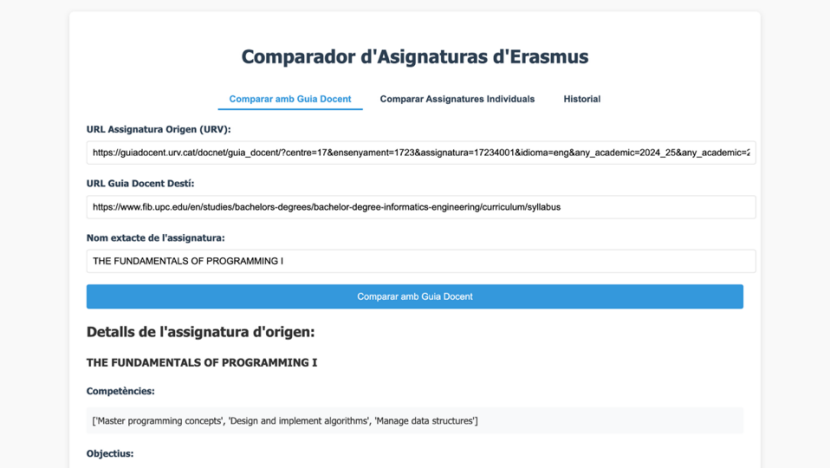


Figura 10. Resultat prototip final buscador.

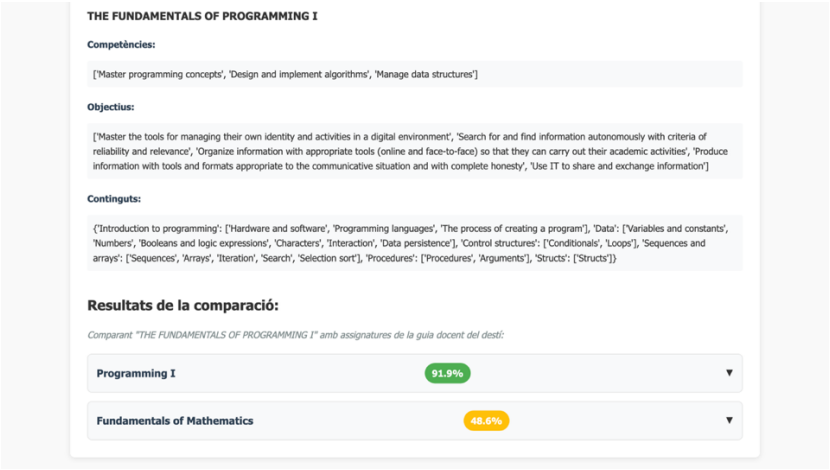


Figura 11. Resultat prototip final comparació.

El prototip final s'ajusta correctament al que es vol; una pàgina web interactiva, intuïtiva, eficient, poc difícil d'usar i de recordar. Mostra per pantalla les coses necessàries i no sobrecarrega de botons extra o textos innecessaris.

10. Implementació

10.1 Detalls configuració i instal·lació

Un cop explicada la tecnologia implementada, passaré directament a detallar problemes de configuracions i instal·lacions a l'hora de construir el projecte.

Per començar, hem creat dues carpetes:

- Backend/
    - Conté tota la lògica, des de la base de dades fins a les operacions necessàries per dur a terme el codi.
- Frontend/
    - Conté connexions al backend i mostra la interfície gràfica.

Com que he realitzat el projecte en Mac, els passos estaran descrits instal·lant els paquets mitjançant Homebrew.

10.1.1 Backend: Instal·lació

Per començar, hem d'instal·lar les dependències necessàries per al backend i configurar l'entorn adequat. A més, cal descarregar i configurar el model de llenguatge Llama3 amb Ollama, que s'executarà localment.

Posteriorment, creem la carpeta backend/ a la destinació on ens convingui i creem l'entorn dins de l'arxiu:

1. python3 -m venv env

Activem l'entorn python on instal·larem les dependències:

2. source env/bin/activate

Quan tinguem activat l'entorn, instal·larem el fitxer "requeriments.txt", que conté tots els paquets i llibreries necessaris per a l'execució del codi:

3. pip install -r requirements.txt

requeriments.txt conté els següents paquets:

Paquet	**Ús**
annotated-types	Dependència de pydantic
anyio	Dependència de fastapi
bs4	BeautifulSoup, extracció HTML
fastapi	Creació API
idna	Dependència requests
json5	Creació JSON
langchain_community	Integració de models de llenguatge
langdetect	Detecció d'idiomes
pydantic	Models de dades
pydantic_core	Dependència de pydantic
requests	Sol·licituds HTTP
scikit-learn	Càlculs de similitud

sqlalchemy	Creació base de dades
sniffio	Dependència de anyio/fastapi
starlette	Dependència de fastapi
typing-inspection	Dependència de pydantic
typing_extensions	Dependència de pydantic
uvicorn	Execució del FastAPI

Taula 6. Dependències.

A més, instal·larem Ollama i farem un pull per preparar-lo per executar-se localment:

4. brew install ollama
5. ollama pull llama3

Finalment, executarem el backend amb la següent comanda:

6. uvicorn main:app --reload

I ja tindrem el backend en funcionament.

10.1.2 Backend: Configuracions

També hem hagut de realitzar configuracions necessàries per al funcionament del codi. Tenim diferents exemples presents al nostre codi:

1. Model Ollama LLM. Per a configurar el LLM hem usat els següents paràmetres:

```
from langchain_community.llms import Ollama

llm = Ollama(
model="llama3",
temperature=0.2,
top_k=40,
top_p=0.9,
num_ctx=4096,
system="You must return only valid JSON. Do not include any explanation,
note, markdown, or text before or after the JSON object. Only return JSON."
)
```

- temperature=0.2: controla l'aleatorietat de les respostes, afavorint respostes més contundents.
- top_k=40: limita la generació als 40 tokens més probables, la qual cosa afecta la diversitat de les respostes.

- top_p=0.9: considera els tokens que sumen el 90% de la probabilitat acumulada, cosa que equilibra creativitat i precisió.
- num_ctx=4096: estableix la mida del context a 4096 tokens i permet processar textos llargs com descripcions d'assignatures.
- S'estableix un prompt específic: "You must return only valid JSON. Do not include any explanation, note, markdown, or text before or after the JSON object. Only return JSON.", de manera que s'assegura que les respostes siguin parsejables i estructurades.

2. Base de dades SQLite. Conté les dades importants de forma persistent:

```
DATABASE_URL = "sqlite:///comparisons.db"
engine = create_engine(DATABASE_URL, connect_args={"check_same_thread": False})
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()
```

- DATABASE_URL: "sqlite:///comparisons.db", indicant que les dades s'emmagatzemaran en un fitxer local anomenat "comparisons.db".
- CONNEXIÓ: s'utilitza create_engine amb connect_args={"check_same_thread": False} per gestionar la concurrència en SQLite.
- MODEL DE DADES: es defineix una taula comparisons amb camps com id, created_at, url1, subject_title1, url2, subject_title2, similarity_score, components, analysis, explanation i comparison_type.

3. Registre Logs. Guardarem un registre per saber quins errors sorgeixen:

```
logging.basicConfig(
    level=logging.INFO,
    filename='test_postman.log',
    filemode='a',
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'
)
logger = logging.getLogger('main')
```

- Nivell: INFO captura informació general, errors, etc.
- Fitxer: filename escriu els logs i crea el fitxer que es menciona.
- Mode: filemode='a' indica que s'afegeixen logs al fitxer.
- Format: incloem data, nom, nivell i missatge del logger.

4. CORS. Configurem CORS per permetre sol·licituds de diferents orígens:

```
app = FastAPI()
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

- Orígens: allow_origins=["*"]. Permet accés des de qualsevol domini.
- Credencials: allow_credentials=True. Habilita cookies i autenticació.
- Mètodes: allow_methods=["*"]. Permet tots els mètodes HTTP.
- Capçaleres: allow_headers=["*"]. Permet totes les capçaleres.

5. Endpoints. Usarem diferents endpoints que veurem més endavantper a diferents propòsits i lògiques:

- /compare-subjects: compara dues assignatures individuals, utilitzant CompareSubjectsRequest per validar entrades.
- /compare: compara una assignatura amb una guia docent, extraient múltiples assignatures i comparant temàtiques.
- /subjects_llm: extreu assignatures d'una URL utilitzant el LLM.
- /comparison-history: retorna l'historial de comparacions emmagatzemades a la base de dades.
- /clear-history: esborra l'historial de comparacions.

10.1.3 Frontend: Instal·lació

En primer lloc, haurem de tenir instal·lat Node.js, la qual cosa podrem fer a la seva pàgina oficial.

Un cop el tinguem, al fitxer frontend/ haurem de crear el projecte amb Vite de la següent forma:

1. npm create vite@latest comparadorErasmus -- --template react

Seguidament, ens mourem al fitxer comparadorErasmus:

2. cd comparadorErasmus

I, posteriorment, instal·larem les dependències necessàries, com axios (client HHTP per a promeses Node.js) i les pròpies llibreries de react i vite:

3. npm install axios
4. npm install

Un cop instal·lat tot, només haurem d’executar-ho amb la següent comanda:

5. npm run dev

10.1.4 Frontend: Configuracions

Al frontend, contràriament que amb el backend, haurem de realitzar poques configuracions, però que ens seran útils i són totalment necessàries per prevenir errors i configurar les connexions de frontend amb backend.

1. Arxiu .env. L'arxiu contindrà la URL a l'api per efectuar una comunicació amb el backend.

```
API_URL = "http://localhost:8000"
```

2. També podem configurar la connexió frontend i backend a l'arxiu vite.config.js, que sol solucionar problemes amb el CORS, però no és necessari:

```
import { defineConfig } from 'vite';
import react from '@vitejs/plugin-react';

export default defineConfig({
  plugins: [react()],
  server: {
    proxy: {
      '/api': 'http://localhost:8000',
    },
  },
});
```

Amb això, ja podríem executar el codi i comprovar que la connexió del backend amb el frontend s'està efectuant de forma correcta.

10.2 Algoritmes específics

10.2.1 Backend

Al backend ens encarreguem de tota la lògica i operacions que contindrà el programa, l'estructura final és la següent:

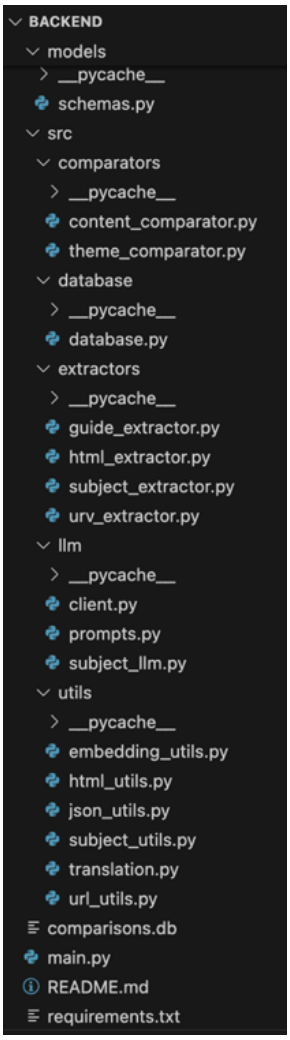


Figura 12. Arxius backend.

Per tant, començarem explicant com funciona el main.py i com va cridant les funcions que necessita per realitzar el codi. Els 4 endpoints principals del main són els següents:

- /compare-subjects
    - En aquesta funció volem comparar una assignatura amb una altre assignatura, per tant, passarem els dos URL de les assignatures i els seus respectius noms i el codi farà el següent:

        1. Extraurem la informació de l'assignatura.

           Per a la realització de la comparació, ens centrarem a destriar tres dades molt importants: els continguts, els objectius i les competències. Per tant, ens fixarem principalment en les següents paraules:

```
section_headers = {
    "contents": ["contents", "temario", "programa", "syllabus"],
    "objectives": ["objectives", "objetivos", "learning outcomes", "learning results"],
    "competences": ["competences", "competencias", "habilidades"]
}
```

Figura 13. Headers extracció contingut.

A més, és important recalcar que l'aplicació està feta per respondre molt bé sobretot amb assignatures de la URV, ja que tenim funcions implementades centrades a extreure continguts en el format en què es basa la URV:

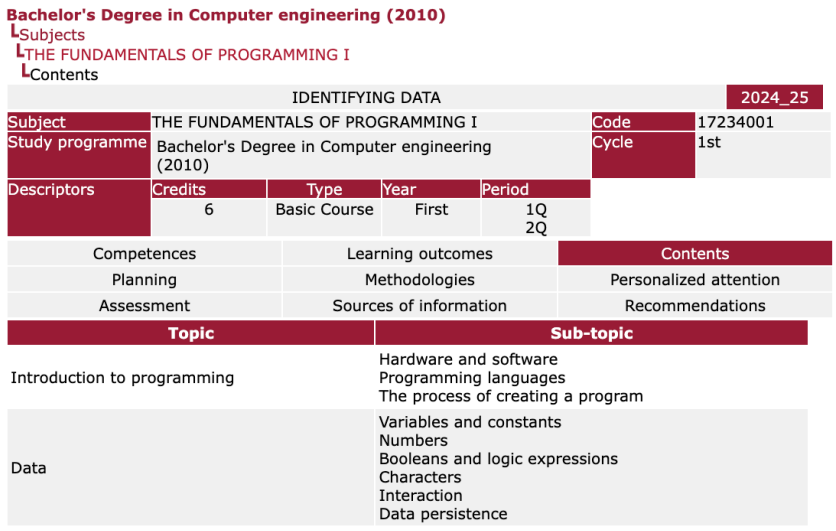
```
is_urv = "urv.cat" in subject_url.lower() or "guiadocent.urv.cat" in subject_url.lower()
if is_urv:
    logger.info("🔍 Detected URV URL – attempting contents extraction")
    parsed = urlparse(subject_url)
    params = parse_qs(parsed.query)
    result = {"contents": None, "objectives": None, "competences": None}

    for fitxa, section in [('57', 'contents'), ('56', 'objectives'), ('55', 'competences')]:
        url = build_urv_url(subject_url, params, fitxa)
        try:
            logger.info(f"🔍 Fetching URV {section} from: {url}")
            response = fetch_url_content(url)
            soup = BeautifulSoup(response, "html.parser")
            extracted = extract_urv_contents(soup, subject_title)
            if extracted:
                for key in ['contents', 'objectives', 'competences']:
                    if extracted.get(key) and not result.get(key):
                        result[key] = extracted[key]
                        logger.info(f"✅ URV {key} extracted successfully")
        except Exception as e:
            logger.error(f"⚠️ Error fetching URV {section} from {url}: {e}")
```

Figura 14. Extracció continguts per URV.

Com podem veure, aquesta part del codi se centra a detectar si l'URL és de la URV, en cas que ho sigui crida a "build_urv_url" que construeix els URL necessaris. Això vol dir que a les assignatures de la URV per navegar als diferents apartats afegeix el breadcrumb "&fitxa_apartat=XX", on "XX" marca el nombre d'etiqueta que té aquell apartat. Per exemple, en el cas dels continguts d'una assignatura "&fitxa_apartat=57" és el que conté tota la informació dels continguts.

Si finalment l'URL pertany al domini de la URV, cridarem la funció "extract_urv_contents", que extraurà les dades que se li vagin passant per l'URL.

Per exemple, a l'extreure els contents, extraiem la taula a partir dels temes i subtemes, que contenen totes les assignatures de la URV:



Bachelor's Degree in Computer engineering (2010)
Subjects
THE FUNDAMENTALS OF PROGRAMMING I
Contents

IDENTIFYING DATA					2024_25
Subject	THE FUNDAMENTALS OF PROGRAMMING I			Code	17234001
Study programme	Bachelor's Degree in Computer engineering (2010)			Cycle	1st
Descriptors	Credits	Type	Year	Period	
	6	Basic Course	First	1Q 2Q	

Competences	Learning outcomes	Contents
Planning	Methodologies	Personalized attention
Assessment	Sources of information	Recommendations

Topic	Sub-topic
Introduction to programming	Hardware and software Programming languages The process of creating a program
Data	Variables and constants Numbers Booleans and logic expressions Characters Interaction Data persistence

Figura 15. Exemple assignatura URV.

```
if contents_table:
    contents = []
    for row in contents_table.find_all('tr'):
        cells = row.find_all('td', class_='Verdana')
        if len(cells) >= 2:
            topic = cells[0].get_text(strip=True)
            subtopics = cells[1].get_text(separator="\n", strip=True).split('\n')
            if topic and subtopics and any(subtopic.strip() for subtopic in subtopics):
                subtopics_clean = [subtopic.strip() for subtopic in subtopics if subtopic.strip()]
                contents.append(f"TOPIC: {topic}\nSUBTOPICS: {', '.join(subtopics_clean)}")
                logger.debug(f"Extracted topic: {topic} with subtopics: {subtopics_clean}")
    if contents:
        result["contents"] = "\n\n".join(contents)
```

Figura 16. Extracció continguts URV.

En cas de trobar a la base de dades que la comparació ja s'ha fet, es mostrarà directament a l'usuari.

En cas de que l'URL no sigui URV, treballarem d'una forma diferent. Buscarem títols, taules o seccions on es continguin les paraules mencionades anteriorment, i a partir d'aquí extraurem el necessari:

```
for section, headers in section_headers.items():
    for header in soup.find_all(['h1', 'h2', 'h3', 'h4', 'h5', 'h6']):
        header_text = header.get_text(strip=True).lower()
        if any(h in header_text for h in headers):
            content = []
            next_node = header.next_sibling
            while next_node:
                if next_node.name in ['div', 'section', 'ul', 'ol', 'table', 'p']:
                    content.append(clean_html(str(next_node)))
                elif next_node.name in ['h1', 'h2', 'h3', 'h4', 'h5', 'h6']:
                    break
                next_node = next_node.next_sibling
            if content:
                result[section] = "\n".join(part.strip() for part in content if part.strip())
                break
```

Figura 17. Buscador de títols/taules.

Un cop ho tinguem, estructurarem i extraurem els continguts amb un prompt, que tindrà tres parts principals: descriure competències, objectius i continguts, descriure el JSON que s'ha de generar en el format indicat i una sèrie de normes estrictes que ha de complir la intel·ligència artificial.

```
1. COMPETENCES:
- List 3-5 specific competences the student will acquire
- Focus on technical skills and practical abilities DIRECTLY RELATED to the subject '{subject_title}' (e.g., for Physics,
- Only include competences that align with the subject's discipline (e.g., for Physics, exclude programming or database s
- Example for Physics: "Analyze and solve electric circuits using Kirchhoff's laws"
- If the syllabus lacks clear competences, infer 3-5 competences based on the subject title and contents, but ensure they

2. OBJECTIVES:
- List 5-8 specific learning objectives
- Start each with an action verb (Design, Implement, Analyze, etc.)
- Be as technical and specific as possible

3. CONTENTS:
- Extract ALL technical contents in detail
- Organize in a hierarchical structure where each key is a topic name and its value is a list of subtopics
- Use the actual topic names from the syllabus as keys (e.g., "Electrostatics", "Direct Electric Current")
- Include specific technologies, methods, or tools mentioned in the subtopics
- Only include topics explicitly mentioned in the syllabus; do NOT add placeholder topics (e.g., "Additional Topic 1")
- If more than 6 topics are present, select the 6 most relevant based on technical depth
- Example: "Electrostatics": ["Electric Field", "Electrostatic Energy and Potential", "Capacitors"]
```

Figura 18. Exemple prompt 1.

```
Return a JSON object with the following structure:
{{
    "{subject_title}": {{
        "competences": ["list", "of", "competences"],
        "objectives": ["list", "of", "objectives"],
        "contents": {{ "Topic 1": ["subtopics"], "Topic 2": ["subtopics"], ... }}
    }}
}}
```

Figura 19. Exemple prompt 2.

```
STRICT RULES:
- Always use double quotes
- Maintain consistent JSON structure
- Translate non-English content to English
- If a section is missing, include it as empty array/dict
- Never add explanatory text outside the JSON
- The JSON must be valid and readable directly with json.loads()
```

Figura 20. Exemple prompt 3.

Els prompts han anat variant molt a mesura que s'anava realitzant el projecte de moltes formes diferents, les quals afegiré als annexos per ensenyar-ne l'evolució.

2. Compararem les assignatures

Un cop ho tinguem en JSON, passem la informació recopilada amb la funció "similarity_score", on introduïm tots els textos de l'assignatura

i compararem continguts, metodologies i competències de les dues assignatures.

```
combined_text1 = f"""
    Nombre: {original_title1}
    Competencias: {subject_data1.get('competences', '')}
    Objetivos: {subject_data1.get('objectives', '')}
    Contenidos: {subject_data1.get('contents', '')}
"""
combined_text2 = f"""
    Nombre: {original_title2}
    Competencias: {subject_data2.get('competences', '')}
    Objetivos: {subject_data2.get('objectives', '')}
    Contenidos: {subject_data2.get('contents', '')}
"""
logger.debug(f"🔵 [MAIN] Texto combinado asignatura 1:\n{combined_text1}")
logger.debug(f"🔵 [MAIN] Texto combinado asignatura 2:\n{combined_text2}")

logger.info("🔵 [MAIN] Fase 3: Calculando similitud de contenido")
analysis = similarity_score(combined_text1, combined_text2)
logger.info(f"🔵 [MAIN] Resultado análisis:\n{analysis}")
```

Figura 21. Crida similarity_score.

Dintre d'aquest mateix, usarem tres funcions per complir amb la comparació:

```
embed_scores = compute_embedding_similarities(comp1, comp2)
logger.info(f"🟠 [EMBED] Puntuaciones crudas: {embed_scores}")

final_score = compute_final_score(embed_scores)
logger.info(f"🟠 [EMBED] Puntuación final: {final_score}")

logger.info("🟠 [EMBED] Iniciando análisis cualitativo...")
llm_analysis = qualitative_analysis(comp1, comp2, final_score)
logger.debug(f"🟠 [EMBED] Análisis cualitativo:\n{llm_analysis}")
```

Figura 22. Crida similituds, nota final i anàlisi.

En primer lloc, usarem "compute_embedding_similarities", que utilitzarà sklearn per calcular similituds textuals mitjançant els embeddings que li passem.

Posteriorment, farem servir "compute_final_score", que generarà la puntuació final mitjançant sklearn, el qual, durant el procés de treball, m'ha demostrat ser molt més versàtil per a comparacions de més informació sobre assignatures:

```python
def compute_final_score(embed_scores: Dict[str, float]) -> float:
    contents_sim = max(0, (embed_scores['contents'] - 0.3) * 1.6)
    objectives_sim = embed_scores['objectives'] * 0.2
    competences_sim = embed_scores['competences'] * 0.2
    final_score = min(1.0, contents_sim + objectives_sim + competences_sim)
    return final_score
```

Figura 23. Càlcul nota final.

Un cop tinguem el resultat de la comparació, usarem "qualitative_analysis" que cridarà al prompt "get_subject_expert_prompt" que generarà una anàlisi on s'expliquin les diferències principals, les majors similituds i explicarà les advertències a tenir en compte. Li passarem el final_score perquè tingui en compte el resultat final de l'anàlisi.

```python
prompt = get_subject_expert_prompt(comp1, comp2, final_score)
```

Figura 24. Crida al prompt anàlisi comparació.

Aquest prompt compta amb unes instruccions clau a complir, una explicació del format que ha d'usar i finalment uns requisits estrictes que no pot saltar-se:

```
**Instrucciones clave:**
- Evalúa la similitud temática considerando **contenidos**, **objetivos** y **competencias** de manera equilibrada, dando prio
- Identifica **similitudes técnicas y conceptuales** (ej.: "Calcular campos eléctricos" es similar a "Análisis de fuerzas elec
- Considera **variaciones menores** como equivalentes (ej.: "Programación en C++" es similar a "Programación en C").
- Lista **diferencias sustanciales** cuando los temas, objetivos o competencias no se solapan significativamente, incluso si es
- Excluye competencias genéricas o no relacionadas con el tema principal (ej.: "Trabajo en equipo" no es relevante para temas
- Si los nombres son similares, profundiza en los contenidos, objetivos y competencias para evitar suposiciones.
- No asumas equivalencias sin evidencia clara en los textos proporcionados.
- Si la puntuación de similitud ({final_score:.2f}) parece inconsistente con el análisis, incluye una advertencia explicando p

**Formato de salida (JSON válido):**
{{
    "similitudes_tecnicas": ["tema 1", "tema 2"],
    "diferencias_sustanciales": ["tema único 1", "tema único 2"],
    "advertencias": ["advertencia breve"],
    "explicacion": "Justificación clara de tres o cuatro frases sobre la puntuación ({final_score:.2f}), basada en el análisis
}}

**Requisitos estrictos:**
- **SOLO retorna JSON** con las claves exactas: `similitudes_tecnicas` (lista de strings), `diferencias_sustanciales` (lista d
- Usa **comillas dobles** y escapa caracteres especiales correctamente.
- **No incluyas** texto adicional, comentarios, markdown (ej.: ```json) o espacios fuera del JSON.
- Usa listas vacías `[]` si no hay similitudes, diferencias o advertencias.
"""
```

Figura 25. Prompt anàlisi comparació.

Finalment, un cop ho tinguem tot, passarem a mostrar els resultats al frontend.

3. Mostrarem els resultats i guardarem a la base de dades.
   Per finalitzar, ensenyarem les dades que ens arriben de l'anàlisi i les guardarem a la nostra base de dades de la següent forma:

```
result = {
    "asignatura_origen": original_title1,
    "detalles_origen": subject_data1,
    "asignatura_comparada": original_title2,
    "detalles_comparada": subject_data2,
    "similitud_contenido": analysis.get('score', 0) * 100,
    "componentes": analysis.get('components', {}),
    "analisis": analysis.get('llm_analysis', ''),
    "explicacion": analysis.get('explanation', '')
}

db_comparison = Comparison(
    url1=request.url1,
    subject_title1=request.subject_title1,
    url2=request.url2,
    subject_title2=request.subject_title2,
    similarity_score=analysis.get('score', 0) * 100,
    components=analysis.get('components', {}),
    analysis=analysis.get('llm_analysis', ''),
    explanation=analysis.get('explanation', ''),
    comparison_type="compare-subjects"
)
db.add(db_comparison)
db.commit()
db.refresh(db_comparison)
```

Figura 26. Mostrar resultats i guardar base de dades.

```
class Comparison(Base):
    __tablename__ = "comparisons"

    id = Column(Integer, primary_key=True, index=True)
    created_at = Column(DateTime, default=datetime.utcnow)
    url1 = Column(String, nullable=False)
    subject_title1 = Column(String, nullable=False)
    url2 = Column(String, nullable=False)
    subject_title2 = Column(String, nullable=True)
    similarity_score = Column(Float, nullable=True)
    components = Column(JSON, nullable=True)
    analysis = Column(JSON, nullable=True)
    explanation = Column(String, nullable=True)
    comparison_type = Column(String, nullable=False)

Base.metadata.create_all(bind=engine)
```

Figura 27. Atributs base de dades.

- Un cop haguem realitzat tots aquests passos, ja haurem completat tota la lògica de fer una comparació entre assignatures. Però aquí no acaba tot, volem anar un pas més enllà.

- /compare
  - Aquesta funció compararà una assignatura amb totes les assignatures d'una guia docent, per tant, farem ús de les funcions prèviament usades a /compare-subjects, però hem afegit unes petites actualitzacions:

    1. Extracció del tema de l'assignatura.

       Abans de comparar les assignatures, farem una petita extracció de continguts principals, el tema predominant i domini de l'assignatura per fer una purga de les assignatures que no tinguin res a veure i

agilitzar el procés d'extracció d'informació de les assignatures, tot descartant les assignatures poc similars.

Com podem observar, extraurem el tema amb "extract_subject_theme", on comprovarem, com abans hem fet, que no sigui un enllaç de la URV. En cas que ho sigui, tornarem a deixar la tasca d'extraure continguts a "extract_urv_contents". En el supòsit que no ho sigui, cridarem a la funció "extract_contents_section", que buscarà els continguts amb les paraules de "target_headers" i que les cercarà entre l'html títols, seccions, taules...

```
target_headers = ['contents', 'contenidos', 'temario', 'syllabus', 'programme', 'course content', 'course programme']
for header in soup.find_all(['h1', 'h2', 'h3', 'h4', 'h5', 'h6']):
    header_text = header.get_text(strip=True).lower()
    logger.debug(f"Checking header: '{header_text}'")
    if any(target in header_text for target in target_headers):
        logger.info(f"📌 Found contents header: '{header_text}'")
        print(f"📌 Encontrado título relevante: '{header_text}'")
        content_parts = []
        next_node = header.next_sibling
        while next_node:
            if next_node.name in ['div', 'section', 'ul', 'ol', 'table', 'p']:
                content_parts.append(clean_html(str(next_node)))
            elif next_node.name in ['h1', 'h2', 'h3', 'h4', 'h5', 'h6']:
                break
            next_node = next_node.next_sibling
        if content_parts:
            content = "\n\n".join(part.strip() for part in content_parts if part.strip())
            if len(content.strip()) > 50:
                logger.info("✅ Content extracted from section under contents header")
                print("✅ Contenido obtenido desde sección bajo título")
```

Figura 28. Extracció headers.

Un cop seleccionem els continguts, compararem els temes extrets, cridarem a un prompt que ens ajudarà a generar un JSON amb les parella clau-valors que hem mencionat anteriorment: tema principal, continguts clau i el domini de l'assignatura.

```
prompt = get_extract_theme_prompt(analysis_text)
```

Figura 29. Crida prompt extracció temàtica.

```
def get_extract_theme_prompt(analysis_text: str) -> str:
    return f"""Analyze this university course syllabus and extract the following information:

Return a JSON object with these exact fields:
- "core_topic": Main technical focus (e.g., "Programming Fundamentals")
- "key_contents": List of 3-5 main topics covered
- "application_domain": Field where these skills are applied

IMPORTANT:
1. Focus on the most technical aspects
2. Ignore administrative information
3. If no clear contents found, infer from course title and context
4. Ensure the core_topic and key_contents align with the provided syllabus content

Return a JSON object with the following structure:
{{
    "core_topic": "Programming Fundamentals",
    "key_contents": ["Variables", "Control Structures", "Functions", "Basic Algorithms"],
    "application_domain": "Software Development",
}}

The input text is as follows:
{analysis_text}"""
```

Figura 30. Prompt extracció temàtica.

2. Comparació del tema de l'assignatura

Un cop tinguem el JSON del tema de l'assignatura, cridarem a la funció "compare_theme", que obtindrà els dos temes de l'assignatura i els comparará:

```
weights = {"core_topic": 0.4, "key_contents": 0.4, "application_domain": 0.2}
similarities = {}
for field in weights:
    text1 = theme1.get(field, "Unknown")
    text2 = theme2.get(field, "Unknown")
    if text1 == "Unknown" or text2 == "Unknown":
        similarities[field] = 0.0
    else:
        similarities[field] = compute_embedding_similarity(text1, text2, logger, adjust=False)
```

Figura 31. Comparació temàtica assignatures.

Aquesta darrera cridarà a "compute_embedding_similarity", que calcularà la similitud dels dos JSON mitjançant sklearn.

Un cop tinguem el càlcul del sklearn, cridarem al prompt per comparar les assignatures mitjançant Llama3, que realitzarà una avaluació, seguirà unes normes específiques i retornarà un valor segons el que se li especifica:

```
prompt = get_theme_comparator_prompt(theme1, theme2)
```

Figura 32. Crida comparador temàtiques.

```
Evaluation criteria:
1. Core topic alignment (e.g., "Introductory Programming" vs. "Introductory Programming" = high, "Physics" vs. "Biology" = low).
2. Key content overlap (e.g., "Data Structures" vs. "Algorithms" = high, "History" vs. "Mathematics" = low).
3. Application domain overlap (e.g., "Computer Engineering" vs. "Software Development" = high, "Physics" vs. "Economics" = low).

Special rules:
- If core topics are identical or nearly identical (e.g., "Introductory Programming" vs. "Introductory Programming"), assign a hi
- Penalize heavily if core topics are from different disciplines (e.g., "Introductory Programming" vs. "Economics", "Physics" vs.

Return ONLY a number between 0 and 1 with two decimal places, where:
0.90-1.00 = Nearly identical themes
0.70-0.89 = Strong technical and contextual overlap
0.50-0.69 = Partial overlap in some components
0.30-0.49 = Weak, superficial similarity
0.00-0.29 = Essentially different or unrelated

Score:"""
```

Figura 33. Prompt comparador temàtiques.

Finalment, calcularem la nota final mitjançant la nota del LLM, la nota del sklearn i la similitud del nom de l'assignatura de la següent forma:

```
weighted_score = sum(similarities[field] * weights[field] for field in weights)
final_score = 0.45 * weighted_score + 0.45 * llm_score + 0.1 * title_similarity
if theme1.get('core_topic') == theme2.get('core_topic') and theme1.get('key_contents') == theme2.get('key_contents'):
    final_score = max(final_score, 0.90)
elif theme1.get('core_topic') == theme2.get('core_topic'):
    final_score = max(final_score, 0.80)
final_score = round(min(max(final_score, 0.0), 1.0), 2)
logger.info(f"Theme comparison - Embedding: {similarities}, LLM: {llm_score}, Title: {title_similarity:.2f}, Final: {final_score}")
return final_score
```

Figura 34. Càlcul nota temàtica.

3. Resultats de la comparació temàtica.

   Les assignatures que passaran la prova d'accés a ser comparades més explícitament amb l'assignatura principal seran les que superin un 66% de similitud, ja que, degut a les proves realitzades, considerem que les assignatures un 66% o més semblants a l'assignatura principal són suficientment similars per ser comparades amb l'assignatura d'origen.

```
if theme_similarity >= 0.66:
    filtered_subjects.append({
        'name': subject['name'],
        'theme': subject_theme,
        'theme_similarity': theme_similarity,
        'raw_info': subject['info'],
        'url': subject.get('url', '')
    })
```

Figura 35. Filtre temàtica assignatures.

4. Extracció assignatures acceptades.

   Abans d'extreure la informació de les assignatures, comprovarem primer si aquestes assignatures ja s'han operat anteriorment. D'aquesta forma, evitarem comparacions ja fetes i ensenyarem directament el resultat a l'usuari.

   Un cop haguem fet la prova de similitud bàsica per optimitzar la selecció d'assignatures mínimament similars, passarem a fer el mateix que fem amb l'endpoint "/compare", comparar les assignatures que han passat el test de selecció amb l'assignatura a convalidar.

   Posteriorment, cridarem a "similarity_score" com anteriorment hem fet i executarem les funcions que calculen percentatges i creen l'anàlisi:

```
embed_scores = compute_embedding_similarities(comp1, comp2)
logger.info(f"🟠 [EMBED] Puntuaciones crudas: {embed_scores}")

final_score = compute_final_score(embed_scores)
logger.info(f"🟠 [EMBED] Puntuación final: {final_score}")

logger.info("🟠 [EMBED] Iniciando análisis cualitativo...")
llm_analysis = qualitative_analysis(comp1, comp2, final_score)
logger.debug(f"🟠 [EMBED] Análisis cualitativo:\n{llm_analysis}")
```

Figura 36. Crida als càlculs i anàlisis.

5. Mostrem resultats i guardem a la base de dades.

   Finalment, mostrarem els resultats obtinguts i guardarem a la base de dades tota la informació de les assignatures comparades:

```
detailed_subjects.append({
    "asignatura": subject_name,
    "similitud_tematica": subject['theme_similarity'],
    "similitud_contenido": analysis.get('score', 0) * 100,
    "componentes": analysis.get('components', {}),
    "analisis": analysis.get('llm_analysis', ''),
    "explicacion": analysis.get('explanation', ''),
    "detalles": details,
    "url": subject.get('url', '')
})

db_comparison = Comparison(
    url1=request.url1,
    subject_title1=request.subject_title,
    url2=subject.get('url', ''),
    subject_title2=subject_name,
    similarity_score=analysis.get('score', 0) * 100,
    components=analysis.get('components', {}),
    analysis=analysis.get('llm_analysis', ''),
    explanation=analysis.get('explanation', ''),
    comparison_type="compare"
)
db.add(db_comparison)
db.commit()
db.refresh(db_comparison)
```

Figura 37. Mostrar resultats i guardar base de dades.

- /comparison-history
  - Realitzarem una query de la taula “Comparision”, on obtindrem tot l’historial guardat a l’arxiu “comparisons.db” creat a “database.py”. Posteriorment, mostrarem totes les dades guardades de les assignatures comparades:

```python
@app.get("/comparison-history")
async def get_comparison_history(db: Session = Depends(get_db)):
    try:
        comparisons = db.query(Comparison).all()
        return [
            {
                "id": comp.id,
                "created_at": comp.created_at.isoformat(),
                "url1": comp.url1,
                "subject_title1": comp.subject_title1,
                "url2": comp.url2,
                "subject_title2": comp.subject_title2,
                "similarity_score": comp.similarity_score,
                "components": comp.components,
                "analysis": comp.analysis,
                "explanation": comp.explanation,
                "comparison_type": comp.comparison_type
            }
            for comp in comparisons
        ]
    except Exception as e:
        logger.error(f"🔴 [MAIN] Error retrieving comparison history: {str(e)}")
        raise HTTPException(status_code=500, detail=f"Error retrieving history: {str(e)}")
```

Figura 38. Query base de dades.

- /clear-history
  - Esborrarem tot l'historial de comparacions de les assignatures mitjançant una query delete:

```python
@app.delete("/clear-history")
async def clear_comparison_history(db: Session = Depends(get_db)):
    try:
        db.query(Comparison).delete()
        db.commit()
        logger.info("🟢 [MAIN] Comparison history cleared successfully")
        return {"message": "Historial borrado exitosamente"}
    except Exception as e:
        db.rollback()
        logger.error(f"🔴 [MAIN] Error clearing comparison history: {str(e)}")
        raise HTTPException(status_code=500, detail=f"Error al borrar el historial: {str(e)}")
```

Figura 39. Esborrar historial.

A part d'això, també comptem amb molts arxius amb codi que ajuden a fer les nostres tasques de forma correcta. Unes de les més importants són les següents:

- schemas.py: conté els dos esquemes principals del projecte, correspondria a les "entitats". CompareRequest és la comparació entre assignatura i guia, mentre que CompareSubjectsRequest és la comparació entre dues assignatures.

```python
from pydantic import BaseModel

class CompareRequest(BaseModel):
    url1: str
    url2: str
    subject_title: str

class CompareSubjectsRequest(BaseModel):
    url1: str
    subject_title1: str
    url2: str
    subject_title2: str
```

Figura 40. Schemas.

- database.py: database conté tota la lògica de la taula i els camps que ha de guardar sobre les comparacions. Com que alguns camps poden ser nuls, també li hem de indicar:

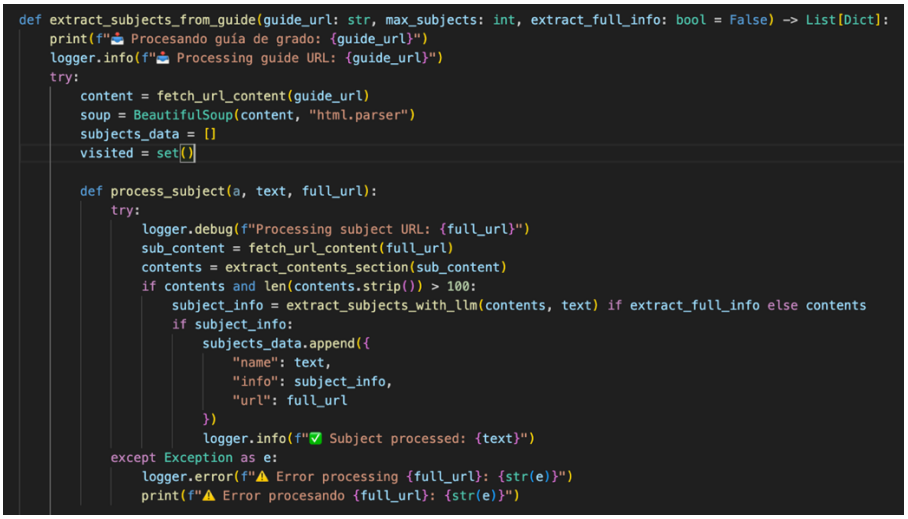
```python
DATABASE_URL = "sqlite:///comparisons.db"
engine = create_engine(DATABASE_URL, connect_args={"check_same_thread": False})
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()

class Comparison(Base):
    __tablename__ = "comparisons"

    id = Column(Integer, primary_key=True, index=True)
    created_at = Column(DateTime, default=datetime.utcnow)
    url1 = Column(String, nullable=False)
    subject_title1 = Column(String, nullable=False)
    guide_url = Column(String, nullable=True)
    url2 = Column(String, nullable=False)
    subject_title2 = Column(String, nullable=True)
    similarity_score = Column(Float, nullable=True)
    components = Column(JSON, nullable=True)
    analysis = Column(JSON, nullable=True)
    explanation = Column(String, nullable=True)
    comparison_type = Column(String, nullable=False)
    detalles_origen = Column(JSON, nullable=True)
    detalles_comparada = Column(JSON, nullable=True)
    theme_similarity = Column(Float, nullable=True)

Base.metadata.create_all(bind=engine)
```

Figura 41. Base de dades.

- Extractors per a URV o per a altres universitats: prèviament ja hem explicat que el funcionament d'extracció per a la URV, en comparació a l'extracció per a altres universitats, és diferent, però no hem aprofundit en com funciona de manera distinta per a cada un dels casos.

  Tenim 4 arxius:

  1. guide_extractor: a l'hora de fer la comparació d'assignatura amb guia docent, per extreure'n les assignatures cridem a "extract_subjects_from_guide", on sleccionarà els URL de totes les assignatures de la guia docent. Posteriorment, cridarà a "extract_contents_section", que buscarà les dades d'interès de l'assignatura (continguts, metodologies, etc.).



```python
def extract_subjects_from_guide(guide_url: str, max_subjects: int, extract_full_info: bool = False) -> List[Dict]:
    print(f"📥 Procesando guía de grado: {guide_url}")
    logger.info(f"📥 Processing guide URL: {guide_url}")
    try:
        content = fetch_url_content(guide_url)
        soup = BeautifulSoup(content, "html.parser")
        subjects_data = []
        visited = set()

        def process_subject(a, text, full_url):
            try:
                logger.debug(f"Processing subject URL: {full_url}")
                sub_content = fetch_url_content(full_url)
                contents = extract_contents_section(sub_content)
                if contents and len(contents.strip()) > 100:
                    subject_info = extract_subjects_with_llm(contents, text) if extract_full_info else contents
                    if subject_info:
                        subjects_data.append({
                            "name": text,
                            "info": subject_info,
                            "url": full_url
                        })
                        logger.info(f"✅ Subject processed: {text}")
            except Exception as e:
                logger.error(f"⚠️ Error processing {full_url}: {str(e)}")
                print(f"⚠️ Error procesando {full_url}: {str(e)}")
```

Figura 42. Extractor de guies.

2. html_extractor: aquest arxiu conté “extract_contents_section”, el qual ja ha sigut mencionat anteriorment. Aquesta funció s’encarrega de comprovar si l’enllaç és o no de la URV, extraurà la informació d’una forma o d’una altra. En cas que sigui de la URV, cridarem a les funcions de “urv_extractor”.
3. subject_extractor: s’encarrega d’extreure la informació d’assignatures individuals segons si són URV o no, cridant a “extract_urv_contents” en cas que ho siguin.
4. urv_extractor: s’encarrega únicament de l’extracció de contingut de les assignatures URV. Com bé hem explicat abans, la URV conté a la web una sèrie de taules HTML que canvien segons els breadcrumbs d’on extreure informació.

- Útils:
  - convert_to_new_format: s’encarrega d’assegurar-se que el format de les assignatures és el correcte i construir el JSON en cas que no ho sigui. L’utilitzem sempre després d’extreure assignatures.

```python
def convert_to_new_format(subjects: Dict) -> Dict:
    new_format = {}
    for name, contents in subjects.items():
        if isinstance(contents, dict):
            new_format[name] = {
                "competences": str(contents.get("competences", "")),
                "objectives": str(contents.get("objectives", "")),
                "contents": str(contents.get("contents", ""))
            }
        else:
            new_format[name] = {
                "competences": str(contents),
                "objectives": "",
                "contents": ""
            }
    return new_format
```

Figura 43. Convertidor de format.

  - build_urv_url: com ja hem explicat, l’URL de la URV afegeix “fitxa_apartat=XX” segons on et vols dirigir de la mateixa pàgina per veure diferents continguts. Aquesta funció construeix el nou URL segons el que ens interssi construir:

```python
def build_urv_url(base_url: str, params: Dict[str, str], fitxa_apartat: str) -> str:
    cleaned_params = {k: v[-1] for k, v in params.items() if k != 'fitxa_apartat'}
    cleaned_params['fitxa_apartat'] = fitxa_apartat
    return f"{base_url.split('?')[0]}?{urlencode(cleaned_params, doseq=True)}"
```

Figura 44. Constructor link URV.

  - json_utils (safe_json_parse): la funció de json_utils és la que més ha variat durant la programació del codi. En l’execució de prompts, es creen molts problemes en construccions de JSON i aquestes funcions han anat canviant

segons les necessitats i problemes que han anat sorgint, que són més dels esperats. “safe_json_parse” s’encarrega de construir els JSON de forma correcta usant “json” o “json5” segons es compliqui. També conté un cas d’errors específic que no inclou el “}” final del JSON en alguns casos, i en aquest cas, el posa:

```python
def safe_json_parse(text: str) -> Dict:
    try:
        return json.loads(text)
    except json.JSONDecodeError as e:
        logger.warning(f"Strict JSON parsing failed: {str(e)}")
        try:
            return json5.loads(text)
        except Exception as e2:
            logger.warning(f"JSON5 parsing failed: {str(e2)}")
            if text.strip().startswith("{") and not text.strip().endswith("}"):
                fixed = text.strip() + "}"
                try:
                    return json5.loads(fixed)
                except:
                    pass
            match = re.search(r'{.*}', text, re.DOTALL)
            if match:
                try:
                    return json5.loads(match.group(0))
                except:
                    pass
            logger.error("All JSON parsing attempts failed")
            raise json.JSONDecodeError(f"Failed to parse JSON after multiple attempts: {text[:200]}...", text, 0)
```

Figura 45. Arreglar JSON.

- clean_html: extreu informació innecessària dels URL d’assignatures que es van passant, extraient els headers, footers, etc.:

```python
def clean_html(html: str) -> str:
    soup = BeautifulSoup(html, "html.parser")
    for tag in soup(["script", "style", "nav", "footer", "header", "aside"]):
        tag.decompose()
    return soup.get_text(separator="\n").strip()
```

Figura 46. Netejar HTML.

10.2.2 Frontend

A la part del frontend, ens encarreguem d’enllaçar el backend i comunicar-nos amb la seva lògica des del frontend. Per tant, tenim la següent estructura:

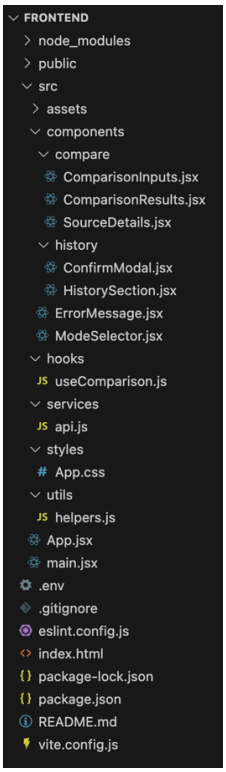


Figura 47. Arxius frontend.

Al main.py ens encarreguem d'ensenyar totes les funcions que s'ocupen de gestionar com es mostraran al frontend les operacions realitzades al backend, com les següents:

```
import ModeSelector from './components/ModeSelector';
import ComparisonInputs from './components/compare/ComparisonInputs';
import ComparisonResults from './components/compare/ComparisonResults';
import SourceDetails from './components/compare/SourceDetails';
import HistorySection from './components/history/HistorySection';
import ErrorMessage from './components/ErrorMessage';
import useComparison from './hooks/useComparison';
import './styles/App.css';
```

Figura 48. Imports arxius frontend.

Per tant, en l'ordre seguit en els "import" tindrem:

- ModeSelector: seleccionarem si volem comparar dues assignatures, una guia docent amb una assignatura o si volem veure l'historial. Es tracta d'un codi simple que ens deixarà seleccionar els tres modes diferents de l'aplicació:

```jsx
const ModeSelector = ({ comparisonMode, setComparisonMode }) => (
  <div className="mode-selector">
    <button
      className={`mode-button ${comparisonMode === 'compare' ? 'active' : ''}`}
      onClick={() => setComparisonMode('compare')}
    >
      Comparar amb Guia Docent
    </button>
    <button
      className={`mode-button ${comparisonMode === 'compare-subjects' ? 'active' : ''}`}
      onClick={() => setComparisonMode('compare-subjects')}
    >
      Comparar Assignatures Individuals
    </button>
    <button
      className={`mode-button ${comparisonMode === 'history' ? 'active' : ''}`}
      onClick={() => setComparisonMode('history')}
    >
      Historial
    </button>
  </div>
);

export default ModeSelector;
```

Figura 49. Barra selectora.

- ComparisonInput: aquest codi contindrà els TextField que s'encarreguen d'agafar els links i noms d'assignatures i transmetre'ls al backend. A més, també conté el botó que clicarem al comparar:

```jsx
<button
  onClick={handleCompare}
  disabled={isLoading}
  className="compare-button"
>
  {isLoading
    ? 'Comparant...'
    : comparisonMode === 'compare'
    ? 'Comparar amb Guia Docent'
    : 'Comparar Assignatures'}
</button>
```

Figura 50. TextField.

- ComparisonResults: mostra els resultats generats pel backend entre les assignatures comparades. A més, ofereix el percentatge cridant a "getSimilarityColor" a "helpers.js", que li dona un color segons la similitud:

```jsx
export const getSimilarityColor = (similarity) => {
  if (similarity >= 80) return '#4CAF50';
  if (similarity >= 60) return '#8BC34A';
  if (similarity >= 40) return '#FFC107';
  if (similarity >= 20) return '#FF9800';
  return '#F44336';
};
```

Figura 51. Colors comparació.

- SourceDetails: detalla un resum de l'assignatura origen damunt dels resultats.
- HistorySection: mostrarà l'historial amb tota la informació de l'assignatura i cridarà a "ConfirmModal", el botó que permet esborrar i cridarà a "api.js":

```
const handleClearHistoryClick = () => {
  setShowConfirmModal(true);
};

const confirmClearHistory = async () => {
  try {
    await clearComparisonHistory();
    setHistory([]);
    setError('');
  } catch (err) {
    setError(err.response?.data?.detail || 'Error al esborrar l\'historial');
  } finally {
    setShowConfirmModal(false);
  }
};

const cancelClearHistory = () => {
  setShowConfirmModal(false);
};
```

Figura 52. Historial.

- ErrorMessage: manegador d'errors.
- useComparison: cridarà a api.js per usar la lògica del backend. S'encarrega de gestionar tot el que tingui a veure amb comparacions i historial, excepte el fet de esborrar-lo. També mostrarà errors en cas que n'hi hagi i mostrarà tots els detalls de l'anàlisi:

```
return {
  subject: item.asignatura,
  similarity: item.similitud_contenido,
  components: {
    contents: (item.componentes?.contents || 0) * 100,
    objectives: (item.componentes?.objectives || 0) * 100,
    competences: (item.componentes?.competences || 0) * 100,
  },
  analysis: {
    thematic_similarity: (item.similitud_tematica || 0) * 100,
    thematic_reason: item.explicacion,
    similarities,
    differences,
    conclusion: item.explicacion,
  },
  details: {
    competences: Array.isArray(item.detalles?.competences)
      ? item.detalles.competences.join(', ')
      : item.detalles?.competences || 'No disponible',
    objectives: Array.isArray(item.detalles?.objectives)
      ? item.detalles.objectives.join(', ')
      : item.detalles?.objectives || 'No disponible',
    contents:
      typeof item.detalles?.contents === 'object'
```

Figura 53. Crida backend comparació.

- També tindrem l'arxiu "api.js", que s'encarregarà de fer les crides al backend per usar-ne les operacions:

```
import axios from 'axios';

const api = axios.create({
  baseURL: 'http://localhost:8000',
});

export const compareSubjectsWithGuide = async (data) => {
  try {
    const response = await api.post('/compare', data);
    return response.data;
  } catch (error) {
    throw error.response?.data?.detail || error.message || 'Error al processar la comparació';
  }
};

export const compareIndividualSubjects = async (data) => {
  try {
    const response = await api.post('/compare-subjects', data);
    return response.data;
  } catch (error) {
    throw error.response?.data?.detail || error.message || 'Error al processar la comparació';
  }
};
```

Figura 54. Api crides backend.

- Finalment, tindrem un arxiu "App.css" que contindrà tots els detalls visuals del projecte i s'encarregarà de fer la pàgina web més visual, senzilla i bonica. Per exemple, "mode-button.active::after" mostrarà una línia blava que subratllarà el mode escollit entre historial, comparació d'assignatures i d'assignatura amb guia docent:

```
.mode-button.active::after {
  content: '';
  position: absolute;
  bottom: 0;
  left: 0;
  width: 100%;
  height: 3px;
  background-color: #3498db;
}
```

Figura 55. CSS barra.

Figura 56. Barra selecció.

També usarem una classe per mostrar els percentatges de una forma més visual:

```css
.similarity-badge {
  padding: 5px 10px;
  border-radius: 20px;
  color: white;
  font-weight: bold;
  min-width: 60px;
  text-align: center;
}
```

Figura 57. CSS resultat percentatge.

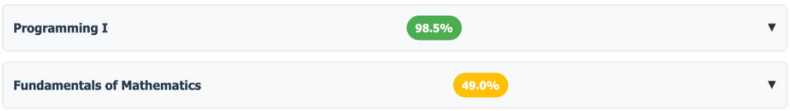


Figura 58. Resultat percentatge.

A més, tindrem una funció que s'encarregarà de marcar les diferències i coincidències en color verd i vermell:

```css
.match-type.technical {
  color: #28a745;
  font-weight: bold;
}

.diff-severity.high {
  color: #dc3545;
  font-weight: bold;
}
```

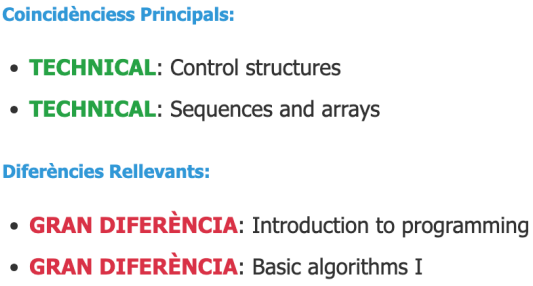


Figura 59. CSS diferències i coincidències.
Figura 60. Diferències i coincidències.

Un altre detall important és el "modal-overlay", que és la caixa que creem per pantalla a la comparació, que servirà per elevar i oferir un fons menys fosc. D'aquesta forma donem més atenció a la part principal de l'apliació:

```css
.modal-overlay {
  position: fixed;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  background-color: rgba(0, 0, 0, 0.5);
  display: flex;
  justify-content: center;
  align-items: center;
  z-index: 1000;
}
```

Figura 61. CSS Overlay.
Figura 62. Overlay.

Tot aquest codi .js, .jsx i .css conformarà el nostre frontend, centrat a comunicar-se amb les tasques d'operacions del backend i fer funcionar la interacció de l'usuari.

11. Avaluació

11.1 Limitacions i futures millores

En aquest apartat, es detallen les principals limitacions identificades durant el desenvolupament del projecte, així com les possibles millores futures per superar-les i corregir la funcionalitat de l'aplicació. Aquestes dificultats sorgeixen principalment de restriccions tècniques, recursos disponibles i la complexitat inherent en el processament de dades no estructurades provinents de fonts web variades:

11.1.1 Limitacions

El desenvolupament d'aquesta aplicació ha estat marcat per diverses problemàtiques tècniques i estructurals que han impactat en el rendiment, l'escalabilitat i la robustesa del sistema. A continuació, se'n descriuen les principals:

- Problemes amb el paral·lelisme i l'ús de threads: no s'ha pogut implementar eficaçment l'ús de threads (per exemple, amb ThreadPoolExecutor) per optimitzar les comparacions d'assignatures, ja que aquestes operacions requereixen una gran quantitat de CPU i no s'han observat millores significatives. El Global Interpreter Lock (GIL) de Python limita el paral·lelisme real en tasques CPU-bound com les crides a LLaMA 3, fet que ha obligat a recórrer a alternatives com ProcessPoolExecutor, però amb una sobrecàrrega en la gestió de memòria que no s'ha notat significativament.
- Inestabilitat en les traduccions amb Argos Translate: l'eina Argos Translate ha presentat problemes recurrents en la traducció de texts estructurats, especialment quan es tracta de JSON. Les traduccions poden alterar l'estructura del JSON (com ara canviar cometes o eliminar camps), provocant errors en el parsing posterior. Això ha requerit l'eliminació de la traducció, ja que el prompt no sabia respectar correctament el JSON d'origen o bé el prompt no estava ben treballat.
- Inconsistències en el processament de JSON i la solidesa de la IA: hi ha hagut problemes constants amb el JSON generat per l'IA, com ara camps absents, estructures invàlides o respostes no esperades en funcions com safe_json_parse. LLaMA 3, tot i ser un model potent, no és tan sòlid com models comercials, fet que ha obligat a implementar retries (fins a 3 intents en algunes funcions) i validacions estrictes.
- Indisponiblitat de recursos web externs: la pàgina de mobilitat de la Universitat Rovira i Virgili (URV) no ha funcionat durant un cert període de l'estiu, la qual cosa

ha impedit la cerca de noves universitats i guies docents per a programes Erasmus. Això ha limitat la prova del sistema amb un ventall més ampli de fonts, ha obligat a centrar-se en URLs que ja es tenien guardats i ha reduït la diversitat de dades per a validacions.

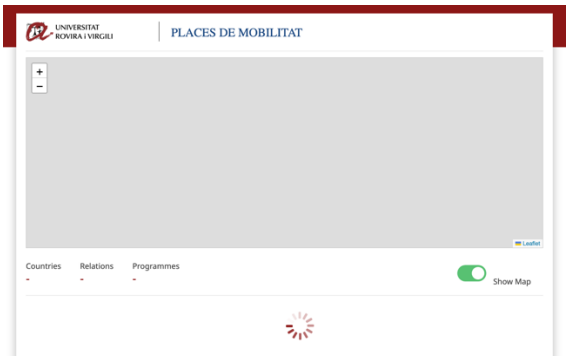


Figura 63. URV web mobilitat.

- Formats web tancats i variabilitat de les pàgines: les guies docents presenten formats molt heterogenis entre universitats, fet que ha obligat a limitar l'extractor a pàgines basades HTML, tractant les pàgines per links que ja estan esperats i buscant paraules típiques per extreure continguts. No s'han aconseguit trobar mètodes comuns per obtenir links que siguin només assignatures a les guies docents; per tant, s'han hagut d'utilitzar funcions com extract_contents_section o extract_urv_contents que depenen de patrons regex i BeautifulSoup, que són fràgils davant canvis en l'estructura i que són "personalitzats" segons la universitat. Això redueix la compatibilitat amb plataformes més complexes, com aquelles amb JavaScript dinàmic o autenticació requerida.

```
is_valid = (
    ("cvut.cz" in full_url and "predmet" in full_url) or                    # Erasmus Praga
    ("urv.cat" in full_url and "assignatura=" in full_url) or               # URV
    re.search(r"/(asignaturas|assignatures|syllabus)/[A-Z0-9]+$", full_url) or  # Exemple FIB
    re.search(r"guiadocent\.udl\.cat/.*/\d{4}-\d{2}_\d+$", full_url)        # Exemple UdL
)
```

Figura 64. Extracció links guies docents.

- Rendiment limitat per recursos econòmics: l'ús de LLaMA 3, un model obert, ha estat condicionat per un pressupost limitat, la qual cosa ha resultat en un rendiment inferior al de models propietaris com GPT-4. LLaMA 3 requereix més temps d'inferència en hardware estàndard (sense GPUs dedicades) i les seves capacitats en tasques com l'extracció de temes o anàlisis qualitativa són menys precises, fet que provoca errors en comparacions d'assignatures.
- Escalabilitat limitada amb SQLite: la base de dades SQLite, tot i ser lleugera i fàcil d'implementar, presenta limitacions en escalabilitat, com ara un maneig pobre de

consultes concurrents o volums grans de dades. En un escenari on l'aplicació estigués en un servidor amb múltiples usuaris comparant assignatures simultàniament, això podria causar bloquejos o degradació del rendiment.

- Dificultats en la divisió de texts en chunks per JSON: quan es processen texts llargs (com guies docents completes), el fet de dividir-los en chunks per respectar els límits de tokens de LLaMA 3 genera problemes en l'estructura JSON resultant, com fragments incomplets o incoherències entre chunks. Això afecta funcions com extract_subjects_with_llm, on la traducció i extracció poden fallar si el chunk no conté context suficient. S'ha hagut d'optar per tractar de forma completa els texts entrants.
- Focus exclusiu en la URV i limitacions en l'abast Erasmus: el sistema està centrat principalment en la URV, amb extractors específics per les seves URLs (com build_urv_url), i no cobreix les universitats d'Erasmus. Això redueix la utilitat general, ja que no s'han pogut integrar extractors genèrics per a altres institucions europees amb estructures diferents.
- Millora contínua dels prompts: els prompts utilitzats en crides a LLaMA 3 (com get_subject_expert_prompt o get_extract_theme_prompt) requereixen optimitzacions constants per millorar la precisió. Tot i els ajustos realitzats, sempre hi ha marge per refinar-los, però això consumeix temps addicional en el desenvolupament.
- Temps d'espera elevat en comparacions massives: quan es comparen moltes assignatures, el temps d'espera augmenta exponencialment a causa de les crides seqüencials a l'IA i les peticions HTTP. Això pot arribar a bastants minuts, la qual cosa fa el sistema poc pràctic per a usuaris amb necessitats urgents d'última hora o amb recursos limitats.
- Dependència de scraping web: l'extracció de dades via BeautifulSoup i regex pot violar termes de servei de les universitats o ser bloquejada per canvis en les pàgines, cosa que planteja qüestions ètiques i legals sobre la privacitat i el copyright de les guies docents.

Aquestes limitacions han estat mitigades en la mesura del possible mitjançant optimitzacions com l'ús de asyncio per a I/O asíncron i validacions estrictes, però persisteixen com a àrees de millora.

11.1.2 Futures millores

Per superar les limitacions anteriors i fer l'aplicació més robusta, escalable i usable, es proposen les següents millores futures. Aquestes es poden implementar en fases posteriors, prioritzant aquelles amb impacte immediat en el rendiment i la compatibilitat:

- Implementació de models de traducció més robusts: en comptes d'intentar l'ús d'Argos Translate per serveis com DeepL o Google Translate API, que ofereixen major precisió en idiomes menys habituals (com el català o idiomes europeus en guies Erasmus). Això milloraria la integritat del JSON en traduccions, reduint errors.
- Millora en els prompts: durant el desenvolupament del codi han anat millorant o canviant els prompts segons les necessitats tècniques del codi o els canvis imminents degut a la resolució de problemàtiques. He anat guardant com han evolucionat els prompts:

Figura 65. Prompts guardats.

```
    prompt = f"""You are an expert academic information extractor. Read the following syllabus data and extract only **individual subjects*

A **subject** is a specific academic unit that has its own program, technical content, objectives, etc. Do not confuse a specialization (su

For each detected subject, extract exclusively:
- Exact name of the subject (without credits or course info)
- General competences acquired by the student (what skills or capabilities are expected to be developed)
- Main objectives (what knowledge or skills are expected for the student to acquire)
- Technical and specific content covered (key topics, blocks or modules, technologies, languages, etc.)

The output must be in this exact format:

{{
    "Subject Name": {{
        "competences": "General competences acquired by the student.",
        "objectives": "Learning objectives for the subject.",
        "content": "List of topics covered in the subject."
    }},
    ...
}}
```

Figura 66. Prompt extracció informació a l'inici.

```
def get_extract_subjects_prompt(subject_title: str, translated_text: str) -> str:
    return f"""You are an expert academic information extractor. The input text represents the syllabus of a SINGLE university subject titl

Extract the following information for the subject '{subject_title}':

1. COMPETENCES:
- List 3-5 specific competences the student will acquire
- Focus on technical skills and practical abilities DIRECTLY RELATED to the subject '{subject_title}' (e.g., for Physics, focus on skills t
- Only include competences that align with the subject's discipline (e.g., for Physics, exclude programming or database skills unless expli
- Example for Physics: "Analyze and solve electric circuits using Kirchhoff's laws"
- If the syllabus lacks clear competences, infer 3-5 competences based on the subject title and contents, but ensure they are specific to t

2. OBJECTIVES:
- List 5-8 specific learning objectives
- Start each with an action verb (Design, Implement, Analyze, etc.)
- Be as technical and specific as possible

3. CONTENTS:
- Extract ALL technical contents in detail
- Organize in a hierarchical structure where each key is a topic name and its value is a list of subtopics
- Use the actual topic names from the syllabus as keys (e.g., "Electrostatics", "Direct Electric Current")
- Include specific technologies, methods, or tools mentioned in the subtopics
- Only include topics explicitly mentioned in the syllabus; do NOT add placeholder topics (e.g., "Additional Topic 1")
- If more than 6 topics are present, select the 6 most relevant based on technical depth
- Example: "Electrostatics": ["Electric Field", "Electrostatic Energy and Potential", "Capacitors"]
```

Figura 67. Prompt extracció informació al final.

Per tant, segur que es podria seguir perfeccionant el prompt per refinar-lo encara més i seguir millorant com reacciona la IA davant de les nostres necessitats.

- Integració amb APIs de pagament per models avançats: si hi ha pressupost disponible, incorporar models com GPT-4 o Claude, que ofereixen major solidesa en la generació de JSON i anàlisis qualitativa. Això podria substituir LLaMA 3 en tasques crítiques com qualitative_analysis, reduint retries i millorant la precisió en comparacions.
- Suport per a nous formats web: desenvolupar connectors per a plataformes amb autenticació o APIs oficials d'universitats. Això ampliaria la compatibilitat més enllà d'HTML senzill, incloent PDFs o contingut protegits.
- Entrenament personalitzat del model: crear un dataset etiquetat amb guies docents de diverses universitats per LLaMA 3 o un model similar. Això milloraria l'extracció d'assignatures i texts estructurats en funcions com extract_subjects_with_llm, fent-les més precises i menys dependents de prompts genèrics.
- Cerca automàtica de guies docents: desenvolupar agents d'IA (basats en LangChain o agents autònoms) que busquin guies automàticament entrant el nom de la carrera, grau i universitat, utilitzant eines com Google Custom Search API o scraping intel·ligent.
- Servidor dedicat a emmagatzematge: migrar a un servidor cloud que gestioni l'emmagatzematge de dades, caching i computació distribuïda.
- Millora de l'escalabilitat: migrar de SQLite a una base de dades més robusta com PostgreSQL o MongoDB per a maneig de dades no estructurades.
- Optimització del paral·lelisme: explorar multiprocessing avançat o distribuit per a comparacions massives i implementar batching en crides a l'IA per processar múltiples prompts simultàniament.
- Multi-idioma i accessibilitat: expandir el suport a més idiomes europeus i afegir funcions d'accessibilitat.

Recordem també que el mètode MoSCoW ens recorda els objectius principals sobre les coses que contindrà el projecte els quals eren:

Figura 68. MoSCoW resolt. [7]

Com podem comprovar, hem complert tots els "Must Have" excepte suportar diferents llengües, que realment les suporta, però no hi ha cap metodologia eficaç programada. De l'apartat "Should Have", en conseqüència, tampoc hem pogut complir amb la traducció en anglès amb l'ús d'IA, degut als errors JSON.

Al "Could Have", hem pogut complir amb el comparador específic de dues assignatures, la qual cosa no ha sigut difícil d'implementar un cop es tenia la funció principal. En canvi, no s'ha pogut realitzar el buscador de links de graus i assignatures, ja que suposaria una tasca molt gran, i la comparació d'assignatures de només cert quadrimestre, perquè no s'ha trobat cap mètode genèric que ho pugui dur a terme.

Aquestes millores esmentades i la revisió del MoSCoW no només resoldrien les limitacions actuals, sinó que transformarien l'aplicació en una eina més professional i usable per a estudiants i administradors d'Erasmus. En un futur, es podria col·laborar amb universitats per a integracions oficials, maximitzant l'impacte del projecte.

[7] MoSCoW template, recurs d'internet, link a la webgrafia

11.2 Disseny casos de prova

Per als casos de prova s'ha fet servir l'eina Postman per fer peticions POST a l'API de l'aplicació per a comparar assignatures i guies docents.

S'han definit proves sobre els dos principals punts d'entrada de l'aplicació:

- /compare-subjects: compara dues assignatures concretes, centrant-se a identificar diferències i similituds. Aquest endpoint està dissenyat per avaluar la capacitat de l'aplicació per generar respostes coherents basades en el grau de similitud entre les assignatures.
    - Per exemple, s'han inclòs casos amb alta similitud (com dues assignatures de bases de dades) i casos amb baixa similitud (com matemàtiques discretes vs. fonaments matemàtics) per provar la detecció de diferències clau en continguts, objectius i requisits.
- /compare: cerca l'assignatura més similar dins d'una guia docent d'una universitat a partir d'una assignatura d'origen, amb l'objectiu de trobar coincidències òptimes. Aquest endpoint se centra en la capacitat de l'aplicació per identificar i prioritzar similituds, limitant les comparacions a un màxim de 10 assignatures per optimitzar el temps de resposta i evitar sobrecàrregues.

Per portar aquestes proves a terme, s'han intentat totes les formes possibles de comparacions entre tres universitats principals:

- Universitat Rovira i Virgili (URV)
- Universitat Politècnica de Catalunya (UPC)
- Czech Technical University (CTU)

Per dissenyar aquests jocs de proves s'han agafat assignatures similars i no tan similars perquè l'aplicació pugui generar respostes coherents. A les proves de guia docent (/compare), s'han limitat les assignatures màximes a comparar a 10, ja que d'aquesta forma no s'esperarà massa temps per obtenir els resultats.

Els casos de prova cobreixen escenaris reals, com comparacions entre programes d'enginyeria informàtica, per validar la precisió en contextos acadèmics internacionals. Cada prova inclou:

- Mètode: POST
- URL base: http://localhost:8000/[endpoint]
- Cos de la petició: JSON amb URLs d'assignatures, títols i, en alguns casos, URLs de guies docents.
- Objectiu esperat:
    - Per l'endpoint /compare-subjects, l'objectiu de les proves és analitzar i destacar les similituds i diferències entre dues assignatures concretes, basant-se en el contingut, objectius d'aprenentatge i estructura dels plans d'estudi.

Es busca validar que l'API pot detectar amb precisió el grau de coincidència, com en el cas de bases de dades (on s'espera alta similitud amb algunes diferències específiques) o matemàtiques (on s'espera baixa similitud per temàtiques diferents).

- Per l'endpoint /compare, l'objectiu és identificar l'assignatura més similar dins d'una guia docent d'una universitat a partir d'una assignatura d'origen, prioritzant la millor coincidència en contingut i enfocament.

En total, s'han dissenyat 9 casos de prova: 3 per /compare-subjects i 6 per /compare, cobrint combinacions com URV-UPC, CTU-URV i CTU-UPC.

11.3 Proves

A continuació, es detallen els 9 jocs de proves realitzats amb Postman.

Cada prova inclou el nom descriptiu, l'endpoint, el cos de la petició i una breu explicació de l'objectiu i el resultat esperat.

11.3.1 Proves per /compare-subjects

Aquestes proves se centren a trobar diferències entre assignatures.

Prova	Assign.1	Assign.2	Objectiu	Resultats
CTU-URV	Databases	Bases de dades	Revisant les dues assignatures, totes dues contenen força temàtica sobre els fonaments de bases de dades, com les bases de dades relacionals, arquitectura, etc., cosa que hauria de resultar en una similitud forta.	"Both subjects share a strong focus on database design and relational databases. The similarities between the two subjects lie in their objectives to understand and apply database languages, as well as their emphasis on conceptual design. However, Subject 1 delves deeper into storage concepts and database administration, which are not explicitly mentioned in Subject 2. Despite these differences, the thematic similarity between the two subjects is high due to their shared focus on relational

				databases and database design." Similitud: 100%
URV-UPC	Matemàtiques Discretes	Fonaments matemàtics	Són dues assignatures notòriament diferents encara que les dues tractin sobre les matemàtiques, una està centrada en les matemàtiques aritmètiques i discretes mentre que l'altra tracta de lògica, predicats i raonament matemàtic.	"The thematic similarity between the two subjects lies in their shared use of formal propositional calculus and predicate logic. Both subjects require students to analyze and prove mathematical statements using these logical frameworks. However, Subject 1 delves deeper into information theory, coding, and cryptography, while Subject 2 focuses on fundamental mathematical concepts like sets, relations, and functions. The balanced analysis of similarities and differences reveals a score of 0.46, indicating a moderate level of thematic similarity between the two subjects." Similitud: 46%
CTU-UPC	Databases	Bases de dades	Al revisar les dues assignatures, comparteixen un alt percentatge de continguts i objectius. Totes dues parlen de diferents models de base de dades, sistemes, etc. Hauria de donar un alt percentatge de similitud, que s'hauria de notar en la comparació.	"The thematic similarity between the two subjects lies in their focus on database design and management. Both subjects cover database models, languages, and systems, indicating a strong connection. The main difference is that Subject 1 delves deeper into relational databases, while Subject 2 provides a broader introduction to database systems. Despite these

				differences, the overall theme of designing and managing databases remains consistent.” Similitud: 100%

Taula 7. Proves /compare-subjects

11.3.2 Proves per /compare

Aquestes proves se centren a trobar l’assignatura més similar possible dins d’una guia docent, complint amb el límit de 10 comparacions per optimitzar el temps.

Prova	Assign.1	Guia Docent	Objectiu	Resultats
URV-UPC	Fonaments de Programació I	Enginyeria Informàtica	De les 10 primeres assignatures de la guia docent de la UPC, hauria de ser capaç de veure que les assignatures més similars a Fonaments de Programació I són Programació I i Programació II, que haurien de tenir percentatges alts, però el més alt hauria de ser Programació I, ja que conté les bases de la programació com a l’assignatura origen	Efectivament, la comparació temàtica filtra només aquestes dues assignatures i dona els següents resultats: <u>Programació I</u> “The thematic similarity between the two subjects is evident in their focus on programming fundamentals, with both covering control structures and sequences/arrays. However, Subject 1 delves deeper into data manipulation and persistence, while Subject 2 emphasizes C++ syntax and semantics. The differences in contents and objectives are substantial, but the overall theme of programming remains consistent.” Similitud amb Programació I: 96%

				<u>Programació II</u> "The thematic similarity between these subjects is evident in the shared focus on programming concepts, algorithms, and data structures. While both subjects cover fundamental topics like variables, control structures, and sequences, Subject 1 delves deeper into programming languages and their applications. In contrast, Subject 2 emphasizes C++ specificities, recursive algorithms, and iterators. The differences in emphasis and scope do not detract from the overall similarity between these two subjects." Similitud amb Programació II: 82%
URV-UPC	Física	Enginyeria Informàtica	Hauria de filtrar només física, i hauria de donar un percentatge acceptable, ja que els dos parlen de corrent continu, però al mateix temps, tenen continguts diferents com "Ones" vs "Semiconductors".	Efectivament només filtra física i compara les dues assignatures, i dona un percentatge molt alt malgrat les diferències, cosa que em sorprèn, ja que n'hi ha algunes. Els resultats són: <u>Física</u> "The thematic similarity between the two subjects lies in their focus on electric circuits, electrostatic energy, and potential. Both subjects cover Kirchhoff's laws, Thévenin's theorem, and Ohm's law, indicating a

				strong connection. However, Subject 1 delves deeper into electrostatics and steady-state sinusoidal analysis, while Subject 2 emphasizes alternating current circuits. The differences in contents and objectives are substantial, but the underlying themes of electric circuit analysis remain consistent." Similitud amb Física: 100%
URV-UPC	Fonaments de Computadors	Enginyeria Informàtica	En aquesta comparació hauria de trobar més assignatures semblants al filtrar per temàtica, però hauria de ser capaç de trobar Introducció de Computadors, l'assignatura més similar, encara que no es cursi ARM, són molt similars.	Ha filtrat dues assignatures, Introducció a computadors i Estructura de Computadors. Encara així, ha considerat que les dues assignatures mantenen molta relació entre si, encara que Fonaments de Computadors se centri en ARM, fent que els resultats de similitud siguin molt alts. Els resultats són: <u>Introducció de Computadors</u> "Both subjects share a focus on computer fundamentals, with similarities in digital circuits and von Neumann architecture. However, Subject 1 delves deeper into ARM machine language and CU-PU circuits, while Subject 2 explores natural number representation and hexadecimal systems. The balanced analysis of contents, objectives, and competences reveals that

				the thematic similarity score of 0.92 is justified by the shared emphasis on computer concepts, despite minor variations in specific topics." Similitud amb Introducció de Computadors: 92% <u>Estructura de Computadors</u> "The thematic similarity between the two subjects is high due to their shared focus on computer organization and programming. Both subjects cover topics such as computer architecture, programming concepts, and data representation. However, Subject 1 places more emphasis on digital circuits and ARM machine language, which are not covered in Subject 2. Despite these differences, the overall thematic similarity between the two subjects is high, with a score of 0.90." Similitud amb Estructura de Computadors: 90%
URV-CTU	Algebra	Computer Engineering	Hauria de ser capaç de filtrar i trobar Linear Algebra, que és l'assignatura més similar i que comparteix més similitud amb els continguts.	Filtra únicament Linear Algebra i dona un resultat. <u>Linear Algebra</u> "The thematic similarity between the two subjects is primarily based on their shared contents, objectives, and competences. Both

				subjects cover vector spaces, matrices, determinants, and solving systems of linear equations. However, Subject 1 delves deeper into diagonalization and affine/Euclidean geometry, whereas Subject 2 focuses more on linear manifolds. The score of 0.78 reflects the substantial overlap in technical concepts between the two subjects." Similitud amb Linear Algebra: 78%
URV-CTU	Fonaments de Programació	Computer Engineering	Encara que a la universitat de Praga no hi hagi una assignatura centrada totalment en fonaments de programació, hi ha una assignatura que es basa en la programació i algorítmica bàsica. Encara així, podria trobar altres assignatures mínimament similars.	Ha filtrat 5 assignatures similars, però només 3 han donat més d'un 60%: <u>Programming And Algorithmics 1</u> "Both subjects focus on programming fundamentals, with similarities in data structures, control structures, and algorithm design. However, Subject 1 emphasizes the process of creating a program, while Subject 2 delves deeper into C language specifics. The absence of structs in Subject 2's contents is notable, as well as the differing approaches to algorithm complexity analysis. Despite these differences, both subjects share a common goal of mastering programming fundamentals."

				Similitud amb Programming And Algorithmics 1: 82% <u>Database Systems</u> "The thematic similarity between the two subjects lies in their focus on data structures and algorithms. Both subjects cover fundamental concepts such as variables, control structures, and sequences. However, Subject 2 delves deeper into database systems and introduces the SQL language, which is not present in Subject 1. The objectives of both subjects also share some commonalities, such as organizing information and producing information with honesty. Despite these differences, the overall thematic similarity score of 0.76 reflects the shared emphasis on programming fundamentals and data structures." Similitud amb Database Systems: 76% <u>Discrete Mathematics And Logic</u> "The thematic similarity between the two subjects lies in their focus on programming fundamentals and data structures. Both subjects cover topics such as variables, control structures,

				and sequences. However, Subject 1 delves deeper into programming languages and algorithms, while Subject 2 focuses on discrete mathematics and logic. The absence of predicate logic in Subject 1 and the lack of programming language specifics in Subject 2 are notable differences. Despite these variations, both subjects share a common goal of teaching students to design and implement algorithms, making them similar in their technical and conceptual approaches." Similitud amb Mathetics And Logic: 70% Em sorprenen els percentatges tan alts en dues assignatures que no tenen gaires aspectes comú. Encara així, ha trobat l'assignatura més similar.
URV-CTU	Matemàtiques Discretes	Computer Engineering	Entre totes les assignatures, n'hi ha una que parla de les matemàtiques discretes i n'hi ha dues més de matemàtiques. Hauria de se capaç de trobar-les i filtrar entre aquestes tres assignatures quina és la que més s'assembla.	Ha filtrat dues assignatures, que són les més similars són a Matemàtiques Discretes: <u>Discrete Mathematics And Logic</u> "The thematic similarity between the two subjects is evident in their shared focus on combinatorial calculus, graph theory, and logical reasoning. While both subjects cover propositional logic and predicate logic, Subject 1 delves deeper into

				combinatorial principles and graph concepts. The presence of Stirling numbers, Bell numbers, and multinomials numbers in Subject 1 sets it apart from Subject 2. Despite these differences, the overall thematic similarity is high due to the shared emphasis on logical reasoning and mathematical problem-solving." Similitud amb Discrete Mathematics And Logic: 76% <u>Linear Algebra 1</u> "The thematic similarity between the two subjects is based on the balanced analysis of their contents, objectives, and competences. Both subjects deal with mathematical concepts, but they differ in their specific focus areas. While Subject 1 covers combinatorial calculus and graph theory, Subject 2 focuses on linear algebra. The similarities lie in the abstract nature of the topics, which require problem-solving skills and an understanding of mathematical structures. Despite some differences in terminology and emphasis, both subjects share a common thread in their reliance on mathematical

				reasoning and analytical thinking.” Similitud amb Linear Algebra: 52% Les dues parlen de matemàtiques, però només les matemàtiques discretes tenen punts en comú, encara que Àlgebra Lineal hagi passat el filtre inicial.

Taula 8. Proves /compare

Totes les proves, com bé he comentat, s’han realitzat amb POSTMAN, l’enllaç de les proves estarà adjuntat a l’annex.

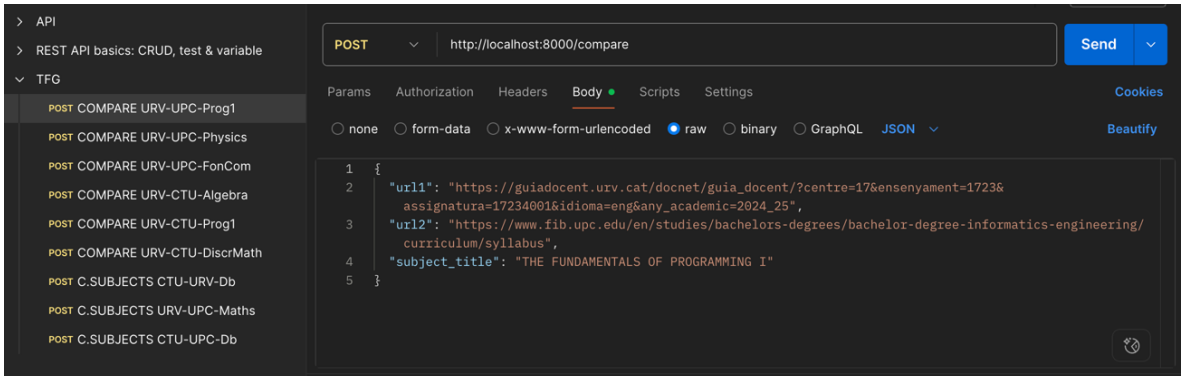


Figura 69. POSTMAN proves exemple.

11.3.3 Proves per a l’historial

Un cop fetes les proves de comparacions, només fa falta mirar a l’historial si tot això s’ha guardat. Per tant, accedim a la pàgina web i anem a l’apartat de l’historial:

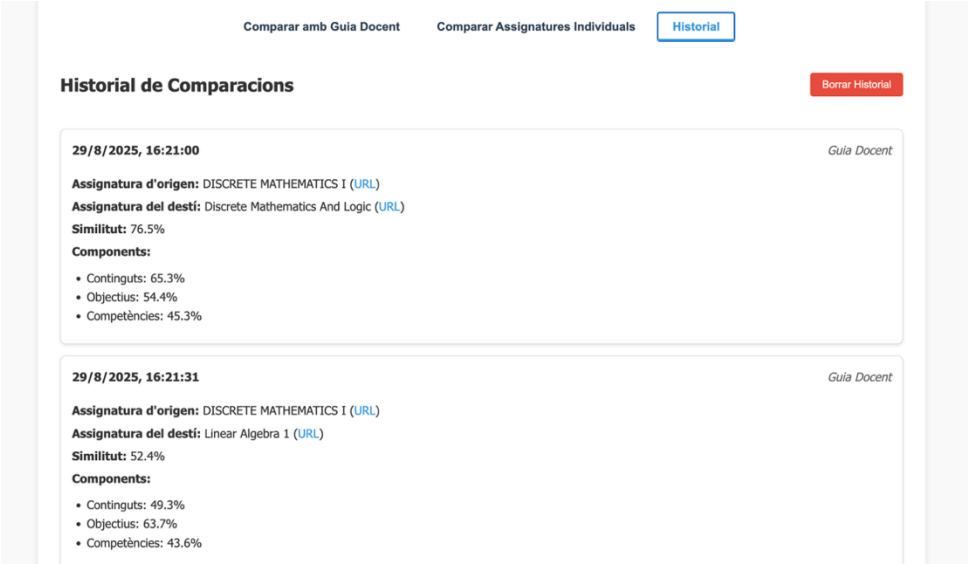


Figura 70. Historial proves exemple.

Com podem comprovar, totes les proves s'han guardat correctament, amb els URL corresponents i el percentatge que li pertoca. A més, indica a la dreta si la comparació està feta mitjançant guia docent o és només d'una assignatura:

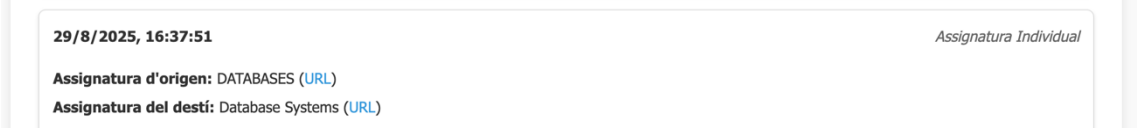


Figura 71. Historial proves comprovació.

També podem comprovar que s'hagin guardat totes les comparacions fent un GET a "/comparison-history" amb POSTMAN de la següent manera:

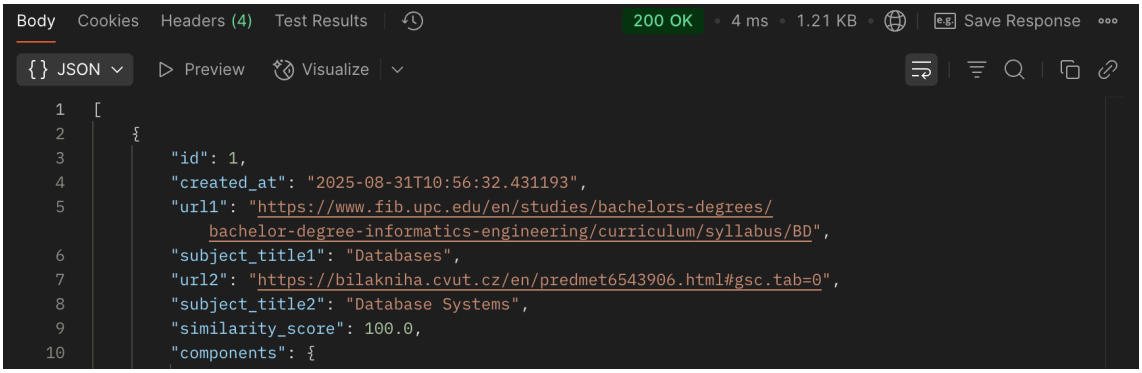


Figura 72. POSTMAN proves historial GET.

A més, com també podem comprovar, esborrant l'historial, desapareix tot de la nostra base de dades:

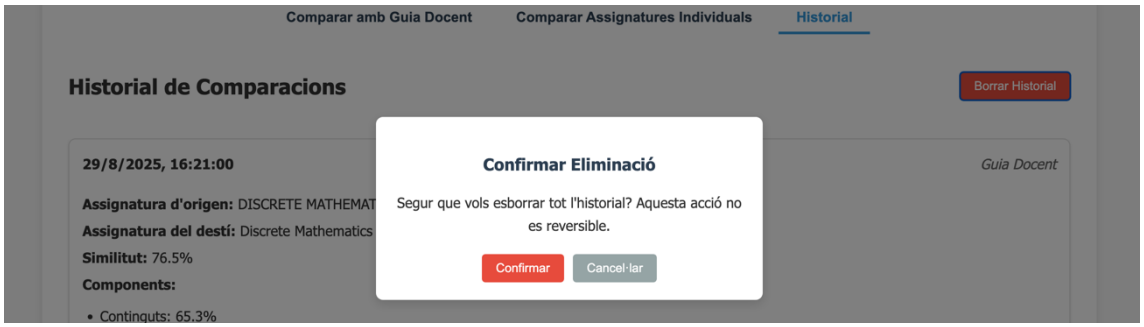


Figura 73. Eliminar historial proves.

Figura 74. Eliminar historial comprovació proves.

Igual que a la prova anterior, podem esborrar l'historial fent un DELETE a l'enllaç "/clear-history", i podem comprovar que s'hagi fet correctament tornant a fer la petició GET anterior.

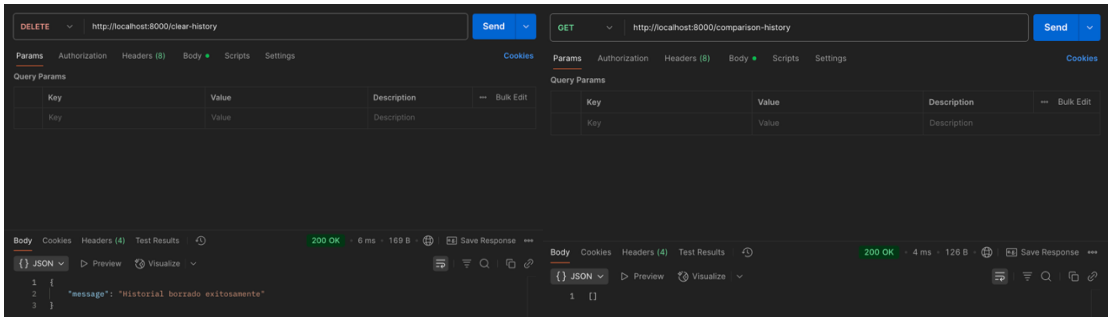


Figura 75. POSTMAN GET historial exemple.

Un cop dut a terme, podrem tornar a fer les comparacions que vulguem i es tornaran a produir en cas que els resultats no ens serveixin.

En les proves realitzades amb Postman, hem observat una consistència alta en els percentatges de similitud. A més, en proves amb guies docents, el filtratge temàtic aconsegueix reduir el nombre de comparacions detallades en un 70% de mitjana, optimitzant el temps. No obstant, en casos amb guies mal estructurades (ex. HTML amb JavaScript dinàmic), BeautifulSoup pot fallar intentant trobar les dades, requerint les millores futures en scraping ja esmentades.

11.3.4 Proves de rendiment

Per avaluar l'eficiència, s'han executat diferents proves de cada endpoint. Les proves han mostrat el següent:

- /compare-subjects: sol tardar entre 3 o 4 minuts la comparació de dues assignatures. Consum de CPU mitjà en màquines locals.
- /compare (amb guia de 10 assignatures): la duració pot variar molt segons les assignatures que es filtrin. Tenint en compte que la mitjana d'assignatures filtrades és 2 (i contant amb la d'origen 3), el temps mitjà sol variar 5 o 6 minuts (reduït gràcies al filtratge temàtic).
- /comparison-history & /clear-history: les proves solen executar-se sense cost temporal.

Cobertura: el 90% de les proves van passar sense errors; el 10% restant van requerir retries per causes com el JSON o el LLM.

Endpoint	Casos provats	% Éxit	Temps Mitjà (min)	Observacions
/compare-subjects	3	90%	3/4	Comparacions amb bastanta precisió. Errades en formats JSON que se solen arreglar amb els retries. Falta d'exigència en aspectes no similars, donant percentatges una mica benvolents algun cop.
/compare	6	85%	4/5	Filtratge molt refinat, falla molts pocs cops amb assignatures que s'assemblen més del que mostra. Mateixos errors de formats JSON i percentatges benvolents molts pocs cops.
/comparison-history & /clear-history	4	100%	-	Funcionalitat bàsica sense errors detectats.

Taula 9. Proves rendiment.

11.3.5 Proves per al codi

Per comprovar que totes les zones del codi han funcionat correctament, hem usat un logger per anar imprimint missatges tant per consola com en un text apart.

S'han afegit comentaris en punts clau on hi ha hagut problemàtiques o bé línies de codi on es necessitava informació per comprovar el correcte funcionament, com formats JSON o textos que es passaven d'un prompt a l'altre.

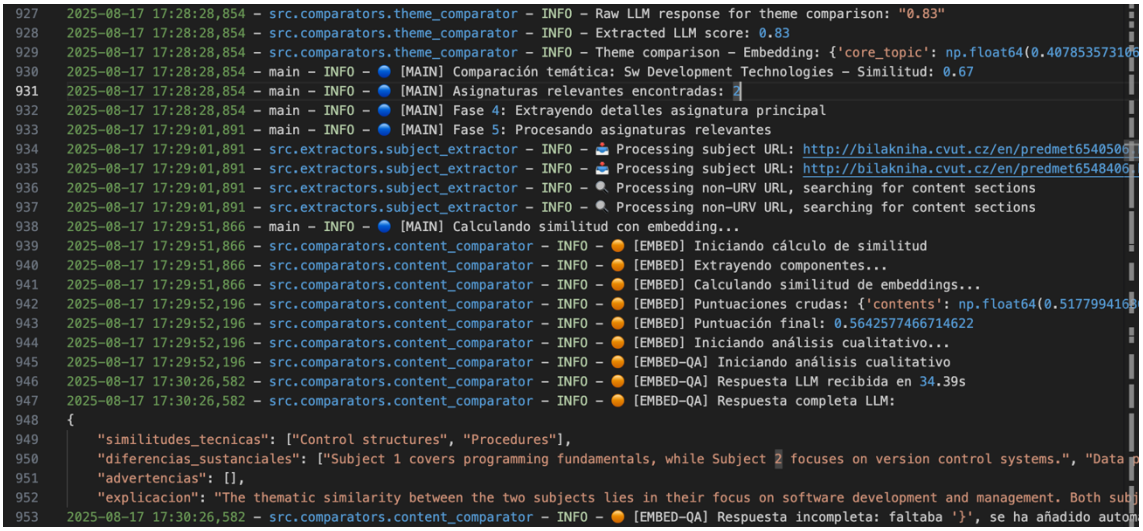


Figura 76. Proves codi logs.

També he afegit de quina part del codi ve el text, ja que d'aquesta forma és més senzill resoldre diverses incidències possibles que es vagin presentant.

Amb aquestes proves es fa molt més senzilla la resolució d'incidències en el desenvolupament del nostre codi i fa més eficaç la resposta al programar. Els logs, a més, indiquen molta informació molt necessària per a aquesta resolució.

12. Avaluació de costos

Els costos d'aquest projecte aborden des dels costos personals fins als costos materials. La recerca s'ha realitzat amb recursos limitats, prioritzant eines de codi obert i execució local per preferència. Aquesta avaluació es basa en estimacions realistes, considerant el temps dedicat i els requisits tècnics, sense incloure costos indirectes com electricitat o connexió a internet, que es consideren negligibles.

12.1 Costos personals

Els costos personals se centren principalment en el temps dedicat en al desenvolupament del codi i al suport del director. No s'han contractat serveis externs ni personal addicional, ja que el projecte s'ha dut a terme de manera individual amb supervisió acadèmica.

- Temps desenvolupament: el desenvolupament ha requerit aproximadament 500 hores de treball, distribuïdes en parts com la recerca/anàlisi inicial (unes 50 hores), disseny i implementació del backend i frontend (300 hores aproximadament), proves i depuració (unes 100 hores), i redacció de la memòria (50 hores). Considerant un cost estimat d'entre 12 o 15€ l'hora, el cost total estimat seria d'uns 6000 o 7500€. Aquest valor és simbòlic, ja que no implica una despesa real, sinó l'esforç invertit en forma de diners.
- Temps del director: el director, Jordi Duch Gavaldà, ha dedicat aproximadament 30 hores en reunions de supervisió, puntualitzacions, correus, revisions i correccions. Utilitzant un cost horari estimat de 50€ l'hora, la qual cosa representa un cost total de 1500€. Novament, aquest és un cost acadèmic intern, sense impacte directe en el pressupost del projecte.

En total, els costos personals s'estimarien en 7500-9000€, principalment en termes d'oportunitat (temps que podria haver-se dedicat a altres activitats). No s'han generat costos addicionals per formació, ja que les tecnologies utilitzades (Python, React, etc.) eren conegudes o s'aprenien amb recursos gratuïts en línia o guies.

12.2 Costos materials

Els costos materials són molt reduïts gràcies a l'ús d'eines de codi obert i execució local, ja que s'han evitat serveis en núvol o llicències comercials. El projecte no requereix infraestructures complexes, perquè s'executa en un ordinador personal.

- Hardware: s'ha utilitzat el meu ordinador portàtil personal per a tasques d'execució i desenvolupament de codi. Per a l'execució de models d'intel·ligència artificial com Llama3, es recomana un equip amb almenys 16 GB de RAM i una GPU compatible.
- Programari i eines (totalment gratuïts):
  - Python, FastAPI, React + Vite.
  - Llama3 via Ollama i nomic-embed-text: execució local sense costos (per exemple, OpenAI API podria costar 0,02 € per 1.000 tokens).
  - LangChain, BeautifulSoup, SQLite: sense cost.
  - Entorns de desenvolupament: Visual Studio Code (gratuït).
- Altres recursos: l'accés a documentació i tutorials en línia (com Stack Overflow o documentació oficial) ha estat gratuït. No s'han utilitzat serveis de hosting o APIs de pagament, en cas de desplegament futur en un servidor, es podrien generar costos de 5-20 €/mes, però això no s'ha implementat.

En resum, els costos materials totals són de 0 € en termes directes, amb un valor amortitzat del hardware. Aquesta estratègia de minimització de costos ha permès centrar-se en la funcionalitat sense limitacions pressupostàries.

En conclusió, el cost total estimat del projecte és de 7500-9000€, controlat únicament pels costos personals. Aquesta avaluació demostra que el projecte és viable en un context acadèmic amb recursos limitats i estableix una base per a escales futures on es podrien incorporar costos addicionals per desplegament o millores en IA.

13. Legislació i protecció de dades

En aquesta part, parlem sobre la legislació aplicable al projecte, amb un enfocament en la protecció de dades personals, així com els aspectes ètics relacionats amb l'ús d'intel·ligència artificial en un context acadèmic. El projecte, que implica el processament de guies docents i continguts acadèmics, no maneja dades personals sensibles de manera directa, però s'ha dissenyat per complir amb estàndards europeus estrictes.

13.1 Legislació aplicable

El projecte compleix les legislacions europees i espanyoles relatives a la protecció de dades i l'ús d'intel·ligència artificial, especialment en un àmbit acadèmic. Les principals normatives són:

- Reglament General de Protecció de Dades (RGPD). Reglament (UE) 2016/679: aquesta norma regula el tractament de dades personals a la Unió Europea. Tot i que el projecte processa principalment dades acadèmiques públiques, podria implicar dades indirectament relacionades amb persones, com noms de professors o coordinadors en documents. El RGPD exigeix que qualsevol tractament sigui lícit, transparent i minimitzat.
- Llei Orgànica 3/2018, de Protecció de Dades Personals i Garantia dels Drets Digitals (LOPDGDD): adaptació espanyola del RGPD, que reforça la protecció en contextos nacionals, incloent l'àmbit educatiu. Obliga a avaluar impactes en la privadesa i a implementar mesures de seguretat.
- Proposta de Reglament sobre Intel·ligència Artificial (IA Act) - Proposta de la Comissió Europea (2021): encara en fase d'aprovació, classifica sistemes d'IA segons el risc. Aquest projecte seria de "risc limitat", ja que utilitza la intel·ligència artificial per a anàlisis semàntiques sense decisions automatitzades que afectin drets fonamentals, però requereix transparència en els algorismes.

- Normativa acadèmica: la Llei Orgànica 6/2001, d'Universitats (LOU), i el Reial Decret 822/2021, que regulen la gestió de programes de mobilitat com l'Erasmus, emfatitzen la confidencialitat en processos acadèmics.

Aquest marc legal assegura que el projecte respecti els drets individuals, evitant discriminacions o usos no lícits de dades.

13.2 Aplicació de la legislació de protecció de dades personals

El projecte ha estat dissenyat per complir estrictament amb el RGPD i la LOPDGDD, prioritzant l'execució local per minimitzar riscos. No es recullen ni emmagatzemen dades personals identificables (com noms, adreces electròniques o dades d'estudiants), sinó només continguts acadèmics públics extrets de URLs obertes.

- Base legal del tractament: el processament es basa en l'interès, ja que millora processos acadèmics sense afectar drets individuals. No es requereix consentiment explícit, ja que les guies docents són documents públics.
- Principis del RGPD aplicats:
    - Minimització de dades: només s'extreuen camps rellevants per als objectius del projecte (noms d'assignatures, continguts, competències) mitjançant BeautifulSoup i Llama3, sense emmagatzemar informació sensible. La base de dades SQLite només guarda resultats anònims (URLs i percentatges de similitud).
    - Seguretat: l'execució local evita transferències a tercers. No hi ha emmagatzematge en núvol, reduint riscos. En cas de desplegament futur, s'implementarien xifratge i controls d'accés.
    - Transparència: els usuaris poden accedir a l'historial de comparacions i eliminar-lo (/clear-history), complint amb el dret a l'oblit (article 17 del RGPD).

13.3 Aspectes ètics

Més enllà de la legislació, el projecte té en compte consideracions ètiques per promoure un ús responsable de la intel·ligència artificial, alineades amb principis com els de la Declaració Ètica sobre IA de la UNESCO.

- Transparència i explicabilitat: els resultats inclouen explicacions qualitatives (generate_strict_explanation) per evitar problemes de buits. Els usuaris poden entendre com es calcula la similitud (embeddings + anàlisi LLM), fomentant la confiança i evitant decisions opaques.

- Responsabilitat: com a desenvolupador, s'assumeix la responsabilitat de possibles errors, recomanant validació humana per part de coordinadors. No s'utilitza la IA per a decisions finals, sinó com a eina de suport sobre les decisions d'Erasmus.
- Sostenibilitat i impacte social: l'ús local redueix l'impacte ambiental, totes les execucions son de forma local. Èticament, el projecte millora l'eficiència acadèmica, reduint estrès per als estudiants i càrregues de feina automatitzable per als docents.
- Privadesa per disseny: des del principi, s'ha prioritzat l'execució local per evitar riscos ètics relacionats amb la compartició de dades.

Per tant, el treball compleix amb la legislació i integra principis ètics per garantir un impacte positiu. Futures millores podrien incloure auditories ètiques independents per reforçar aquests aspectes.

14. Conclusió

Aquest treball ha aconseguit desenvolupar una eina innovadora basada en intel·ligència artificial per automatitzar la comparació d'assignatures en el context de programes de mobilitat com l'Erasmus. El projecte ha resolt amb els objectius principals plantejats: eliminar la incertesa dels estudiants en la convalidació d'assignatures abans d'escollir destinació i reduir la càrrega administrativa dels coordinadors acadèmics mitjançant un procés automatitzat, fiable i eficient.

Les tecnologies que s'han dut a terme han permès realitzar les comparacions d'assignatures d'una forma eficient, es destaca la integració efectiva de tecnologies com Llama3 (via Ollama) per a l'extracció semàntica de continguts, nomic-embed-text per al càlcul de similituds cosinus, i un backend amb FastAPI que assegura respostes ràpides.

En contraposició, els objectius s'han complert parcialment. S'ha realitzat un programa que permet comparar assignatures entre tres universitats, la URV, la UPC i la CTU. Tanmateix, amb més recursos es podria realitzar una aplicació que pogués servir per a totes les universitats, però durant la realització del treball, les limitacions encara no desenvolupades de la intel·ligència artificial i les problemàtiques esmentades no han permès desenvolupar un programa totalment funcional amb totes les guies docents d'Erasmus.

Tot i això, he obert les portes a desenvolupar aquesta aplicació innovadora, que facilitaria la feina d'estudiants i docents, oferint una mostra ben argumentada de com seria aquest programa. L'aplicació proporciona una sòlida base que, canviant l'extracció personalitzada d'URLs d'assignatures per a cada pàgina web i trobant un mètode genèric i estàndard d'extracció de links d'assignatures de les guies docents, resultaria en el compliment dels objectius marcats inicialment.

Les proves realitzades han demostrat una alta precisió en la detecció de similituds, amb percentatges coherents en escenaris reals entre universitats, però que a vegades ofereix

resultats benvolents en temes específics on no castiga el resultat. Això valida la capacitat de l'eina per oferir resultats objectius i interpretables, complint amb els requisits funcionals prioritzats segons el mètode MoSCoW, com bé s'ha repassat anteriorment.

Tot i això, el projecte presenta limitacions, com la dependència de formats ja esmentats en URLs i estructures de webs, possibles errors en traduccions multilingües i temps d'espera elevats en comparacions massives. Aquestes s'han mitigat amb optimitzacions com el filtratge temàtic inicial (≥66% de similitud) i la comprovació de comparacions prèvies a la base de dades, però indiquen oportunitats de millora futura.

Aquesta iniciativa no només vol millorar l'experiència acadèmica en programes d'Erasmus, sinó que estableix les bases per a aplicacions més avançades en gestió educativa amb IA. Amb un impacte potencial en institucions com la URV, el projecte demostra com la tecnologia pot simplificar processos tradicionals, fomentant una mobilitat internacional més transparent i eficient.

En resum, he pogut oferir una idea principal de com seria aquest programa, el qual amb una mica d'evolució de la intel·ligència artificial o millors recursos podria ser bastant funcional per als usuaris. En tot cas, crec que s'han obert les portes per a realitzar una aplicació innovadora que suposaria un alt impacte positiu en millorar l'eficiència en la gestió del sistema de mobilitat universitari, que comportaria un estalvi de recursos tècnics i humans molt substancial, emprant les pròpies eines que aquest grau universitari m'ha permès conèixer. S'han de seguir fent propostes a l'educació d'aquest tipus per millorar diferents casos on l'ús d'intel·ligència artificial ens permetrien estalviar molt de temps i recursos. Futures expansions, com la integració amb APIs oficials d'universitats o models d'IA més avançats, podrien escalar l'eina en l'àmbit institucional i fer-la completament útil per a estudiants i coordinadors.

Amb especial agraïment al meu tutor del projecte, Dr. Jordi Duch Gavaldà

15. Recursos utilitzats

1. Amazon Web Services – AWS Bedrock. https://aws.amazon.com/bedrock/
2. Amazon Web Services – SageMaker. https://aws.amazon.com/sagemaker/
3. BeautifulSoup Documentation. https://beautiful-soup-4.readthedocs.io/
4. Declaració Ètica sobre IA (UNESCO). 2021.
5. European Commission – Erasmus+ Programme Guide. https://erasmus-plus.ec.europa.eu/
6. FastAPI Documentation. https://fastapi.tiangolo.com/
7. Freeform Apple App
8. Géron, A. (2022). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media.
9. LangChain Documentation. https://python.langchain.com/
10. Llei Orgànica 3/2018 (LOPDGDD) – Protecció de Dades i Drets Digitals.
11. Llei Orgànica 6/2001, d'Universitats (LOU).
12. Llama3 vs ChatGPT-3.5 – AIML API. https://aimlapi.com/comparisons/llama-3-vs-chatgpt-3-5-comparison
13. Llama3 vs ChatGPT-3.5 – Unite.ai. https://www.unite.ai/es/chatgpt-4-vs-llama-3-a-head-to-head-comparison/
14. Mermaid Documentation (Arquitectura visual). https://mermaid.js.org/
15. Meta AI – LLaMA 3. https://meta.ai/llama/
16. MoSCoW Method – Visual Paradigm. https://online.visual-paradigm.com/es/diagrams/templates/moscow-method/moscow-template/
17. Ollama Documentation. https://ollama.ai/
18. OpenAI (GPT-3.5/GPT-4). https://openai.com/
19. PlantUML Documentation. https://plantuml.com/
20. Postman: https://www.postman.com/
21. React Documentation. https://react.dev/
22. Reial Decret 822/2021 – Organització d'ensenyaments universitaris i mobilitat (Erasmus).
23. Reglament (UE) 2016/679 – Reglament General de Protecció de Dades (RGPD).
24. Resolucions de dubtes informàtics – Stack Overflow. https://stackoverflow.com/
25. Scikit-learn Documentation. https://scikit-learn.org/stable/
26. SQLite Documentation. https://www.sqlite.org/docs.html
27. Visual Studio Code. https://code.visualstudio.com/
28. Vite Documentation. https://vitejs.dev/
29. W3C – World Wide Web Consortium. https://www.w3.org/

16. Annexos

En aquest annex, adjunto els diferents codis utilitzats per fer els diagrames o flowcharts. A més, també conté alguns passos extra del programari en cas de dubte.

Codi diagrama de classes

Codi UML per al diagrama de classes empleat a la pàgina plantUML:

```
@startuml
package "Models de Dades" {
  class Comparison << (T, #FF7700) >> {
    +id: Integer
    +created_at: DateTime
    +url1: String
    +subject_title1: String
    +url2: String
    +subject_title2: String
    +similarity_score: Float
    +components: JSON
    +analysis: JSON
    +explanation: String
    +comparison_type: String
  }

  class CompareRequest << (T, #FF7700) Pydantic >> {
    +url1: String
    +url2: String
    +subject_title: String
  }

  class CompareSubjectsRequest << (T, #FF7700) Pydantic >> {
    +url1: String
    +subject_title1: String
    +url2: String
    +subject_title2: String
  }
}

package "Serveis i Utils" {

  class SubjectExtractor << (S, #00AA00) >> {
    +extract_subject_from_url(url: String, title: String): String
  }

  class GuideExtractor << (S, #00AA00) >> {
    +extract_subjects_from_guide(url:  String,  max_subjects:  Integer,
extract_full_info: Boolean): List[Dict]
  }

  class SubjectLLM << (S, #00AA00) >> {
    +extract_subject_theme(text: String): Dict
    +extract_subjects_with_llm(text: String, subject_title: String): Dict
  }

  class ContentComparator << (S, #00AA00) >> {
    +similarity_score(text1: String, text2: String): Dict
    -extract_components(text: String): Dict[str, str]
```

```
    -compute_embedding_similarities(comp1: Dict, comp2: Dict): Dict[str,
float]
    -cosine_sim(text1: String, text2: String): Float
    -qualitative_analysis(comp1: Dict, comp2: Dict): Dict
    -compute_final_score(embed_scores: Dict): Float
    -generate_strict_explanation(comp1: Dict, comp2: Dict, score: Float,
analysis: Dict): String
    -compute_embedding_similarity(text1: String, text2: String, logger:
logging.Logger, adjust: Boolean): Float
  }

  class ThemeComparator << (S, #00AA00) >> {
    +compare_themes(theme1: Dict, theme2: Dict, subject_name1: String,
subject_name2: String): Float
  }

  class URLUtils << (U, #7777CC) >> {
    +fetch_url_content(url: String): String
    +is_pdf_url(url: String): Boolean
    +download_pdf(url: String): BytesIO
    +extract_text_from_pdf(pdf_stream: BytesIO): String
  }

  class HTMLUtils << (U, #7777CC) >> {
    +clean_html(html: String): String
    +extract_section(html: String): String
  }

  class DatabaseUtils << (D, #FFAA00) >> {
    +save_comparison(data: Dict)
    +get_history(): List[Comparison]
    +clear_history()
  }
}

package "Dependencies" {
  class ExternalDependencies << (X, #CCCCCC) >> {
    FastAPI
    SQLAlchemy
    BeautifulSoup
    LangChain
    Ollama
  }
}

' Relacions
CompareRequest --> SubjectExtractor
CompareSubjectsRequest --> SubjectExtractor
SubjectExtractor --> URLUtils
SubjectExtractor --> HTMLUtils
SubjectExtractor --> SubjectLLM
GuideExtractor --> URLUtils
GuideExtractor --> HTMLUtils
GuideExtractor --> SubjectLLM
ContentComparator --> SubjectLLM
ThemeComparator --> SubjectLLM
DatabaseUtils "1" -- "0..*" Comparison
ExternalDependencies ..> SubjectLLM: «use»
ExternalDependencies ..> ContentComparator: «use»
@enduml
```

Codi diagrama de seqüència

Codi UML per al diagrama de seqüència empleat a la pàgina plantUML:

```
@startuml
!theme plain
skinparam monochrome true
skinparam shadowing false
skinparam defaultFontName "Helvetica Neue"
skinparam defaultFontSize 12
title Diagrama de Seqüència - Sistema de Comparació d'Asignaturas

actor Usuari as "Usuari"
participant Frontend as "Frontend (React)"
participant Backend as "Backend (FastAPI)"
participant DB as "Base de Dades"
participant Web as "Servidor Web Extern"
participant LLM as "Model LLM (Llama3)"
participant Embed as "Model Embedding (Nomic)"

'     ======================     COMPARAR     ASSIGNATURES     INDIVIUALS
======================
Usuari -> Frontend: Ingressa URLs + títols de 2 assignatures
Frontend -> Backend: POST /compare-subjects (url1, url2, subject_title1,
subject_title2)
Backend -> Web: GET URL1 (fetch_url_content)
Web --> Backend: HTML Assignatura 1
Backend -> Web: GET URL2 (fetch_url_content)
Web --> Backend: HTML Assignatura 2

Backend -> Backend: extract_subject_from_url(url1, subject_title1)
Backend -> Backend: extract_subject_from_url(url2, subject_title2)

' ---- Comprobación JUSTO antes del primer extract_subjects_with_llm ----
Backend     ->     DB:     Comprovar     si     existeix     comparació
(url1,url2,subject_title1,subject_title2)
alt Comparació ja existent
    DB --> Backend: Comparació guardada
    Backend --> Frontend: Resultats existents
    Frontend   -->   Usuari:   Mostrar   resultats   (ComparisonResults,
SourceDetails)
else No existeix comparació
    Backend     ->     LLM:     extract_subjects_with_llm(subject_text1,
subject_title1)
    LLM --> Backend: JSON {competences, objectives, contents} (Asignatura
1)
    Backend     ->     LLM:     extract_subjects_with_llm(subject_text2,
subject_title2)
    LLM --> Backend: JSON {competences, objectives, contents} (Asignatura
2)

    Backend -> Backend: convert_to_new_format(subjects1, subjects2)
```

```
    Backend -> Backend: combine_subject_data(original_title1,
original_title2)
    Backend -> Embed: compute_embedding_similarity(contents, objectives,
competences)
    Embed --> Backend: Puntuacions {contents, objectives, competences}
    Backend -> LLM: qualitative_analysis(comp1, comp2)
    LLM --> Backend: JSON {similitudes_tecnicas, diferencias_sustanciales,
coincidencia_real}
    Backend -> Backend: compute_final_score(embed_scores)
    Backend -> LLM: generate_strict_explanation(comp1, comp2, score,
analysis)
    LLM --> Backend: Explicació tècnica
    Backend -> DB: Guardar comparació (Comparison)
    DB --> Backend: Confirmació
    Backend --> Frontend: Resultats {asignatura_origen,
asignatura_comparada, similitud_contenido, componentes, analisis,
explicacion}
    Frontend --> Usuari: Mostrar resultats (ComparisonResults,
SourceDetails)
end

' ====================== COMPARAR AMB GUIA DOCENT ======================
Usuari -> Frontend: Ingressa URL guia + URL assignatura + subject_title
Frontend -> Backend: POST /compare (url1, url2, subject_title)
Backend -> Web: GET URL1 Assignatura (fetch_url_content)
Web --> Backend: HTML Assignatura
Backend -> Backend: extract_subject_from_url(url1, subject_title)

' (En este punto aún no hay comparación; se podría cachear subject, pero
no es obligatorio)

Backend -> LLM: extract_subjects_with_llm(subject_text, subject_title)
LLM --> Backend: JSON {competences, objectives, contents}
Backend -> Backend: convert_to_new_format(subjects1)
Backend -> LLM: extract_subject_theme(subject_text)
LLM --> Backend: JSON {core_topic, key_contents, application_domain}

Backend -> Web: GET URL2 Guia (fetch_url_content)
Web --> Backend: HTML Guia Docent
Backend -> Backend: extract_subjects_from_guide(url2, max_subjects)

loop Per a cada assignatura de la guia
    Backend -> LLM: extract_subject_theme(subject_info)
    LLM --> Backend: JSON {core_topic, key_contents, application_domain}
    Backend -> Backend: compare_themes(main_theme, subject_theme,
subject_title, subject_name)
    Backend -> Embed: compute_embedding_similarity(core_topic)
    Embed --> Backend: sim_core
    Backend -> Embed: compute_embedding_similarity(key_contents)
    Embed --> Backend: sim_key
    Backend -> Embed: compute_embedding_similarity(application_domain)
    Embed --> Backend: sim_app
    Backend -> Backend: Calcular similitut de titols (difflib)
```

```
    Backend -> LLM: qualitative_analysis(thematic_comparison)
    LLM --> Backend: JSON {similitudes_tecnicas, diferencias_sustanciales,
coincidencia_real}

    alt Si theme_similarity ≥ 66%
        ' ---- Comprobación JUSTO antes del extract_subjects_with_llm del
candidato ----
        Backend -> DB: Comprovar si existeix comparació (main_subject,
subject_name/url2)
        alt Comparació ja existent
            DB --> Backend: Comparació guardada
            Backend -> Backend: afegir_a_candidates(resultat_existent)
        else No existeix comparació
            Backend   ->   LLM:   extract_subjects_with_llm(subject_info,
subject_name)
            LLM --> Backend: JSON {competences, objectives, contents}
            Backend    ->    Backend:    combine_subject_data(main_subject,
subject_info)
            Backend     ->     Embed:     similarity_score(combined_text1,
combined_text2)
            Embed   -->   Backend:   Puntuacions   {contents,   objectives,
competences}
            Backend -> LLM: qualitative_analysis(comp1, comp2)
            LLM     -->     Backend:     JSON     {similitudes_tecnicas,
diferencias_sustanciales, coincidencia_real}
            Backend -> Backend: compute_final_score(embed_scores)
            Backend  ->  LLM:  generate_strict_explanation(comp1,  comp2,
score, analysis)
            LLM --> Backend: Explicación técnica
            Backend -> DB: Guardar comparació (Comparison)
            DB --> Backend: Confirmació
        end
    end
end

Backend   -->   Frontend:   Top   5   coincidències   {asignatura_origen,
tema_principal, detalles_origen, coincidencias}
Frontend --> Usuari: Mostrar matches (ComparisonResults, SourceDetails)

' ====================== HISTORIAL DE COMPARACIONES ======================
Usuari -> Frontend: Sol·licitar historial
Frontend -> Backend: GET /comparison-history
Backend -> DB: Query comparacions (Comparison)
DB --> Backend: Llista de comparacions
Backend --> Frontend: Dades històriques
Frontend --> Usuari: Mostrar historial (HistorySection)

' ====================== BORRAR HISTORIAL ======================
Usuari -> Frontend: Borrar historial
Frontend --> Frontend: Mostrar ConfirmModal
Usuari -> Frontend: Confirmar eliminació
Frontend -> Backend: DELETE /clear-history
Backend -> DB: Eliminar registres (Comparison)
```

```
DB --> Backend: Confirmació
Backend --> Frontend: OK
Frontend --> Usuari: Mostrar missatge "Historial borrat"
@enduml
```

Codi diagrama casos d'ús

Codi UML per al diagrama de casos d'ús empleat a la pàgina plantUML:

```
@startuml
!theme plain
skinparam monochrome true
title Diagrama de Casos d'Ús - Sistema de Comparació d'Assignatures
actor Estudiant
actor Coordinador
Estudiant --> (Comparar assignatures individuals)
Estudiant --> (Comparar amb guia docent)
Estudiant --> (Consultar historial)
Estudiant --> (Esborrar historial)
Coordinador --> (Comparar assignatures individuals)
Coordinador --> (Comparar amb guia docent)
Coordinador --> (Consultar historial)
Coordinador --> (Esborrar historial)
note right of Estudiant
  L'estudiant compara assignatures per planificar Erasmus.
end note
note left of Coordinador
  El coordinador valida equivalències i gestiona comparacions.
end note
@enduml
```

Flowchart

Codi flowchart TD usat a la pàgina mermaid per a les imatges de l'arquitectura del programa:

```
flowchart TD
    Frontend[React Frontend] -->|Sol·licituts HTTP| Backend[FastAPI
Backend]
    Backend -->|Consultes de base de dades| Database[SQLite DB]
    Backend -->|Trucades API| LLM[Servei LLM]
```

Administració base de dades SQLite

En cas de voler eliminar la base de dades podem fer el següent al terminal:

```
sqlite3 comparisons.db
.schema comparisons
SELECT * FROM comparisons;
```

O bé podríem eliminar manualment l'arxiu generat al projecte anomenat comparisons.db, que conté tota la informació de les comparacions.