UNIVERSITAT DE BARCELONA

Facultat de Matemàtiques
i Informàtica

Treball final de grau

DOBLE TITULACIÓ DE MATEMÀTIQUES I ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

Agent Educatiu Multiassignatura: Disseny i Implementació amb Rasa i Docker

---

Autor: Pau Soler Valadés

Directors: Dra. Inmaculada Rodríguez Santiago
Dra. Susana Romano Rodríguez
Realitzat a: Dept. Matemàtiques i Informàtica

Barcelona, 13 de juny de 2023

Abstract

Nowadays, general interest on chatbots or Conversational Agents has augmented drastically due to the introduction of the Large Language Models (LLMs) empowering ChatGPT and Google Bard. Despite the high intellectual capacities and use of natural language, LLMs suffer defects such information deliriums and loss of the context window of the conversation, making them unreliable for specific tasks in some controled environments.

This project presents a Conversational Agent provided by Rasa and enhanced with calls to OpenAI's *GPT-3.5-turbo* model to perform specific NLG (Natural Language Generation task) complementing the Rasa intents-driven design. Project context is a multi-subject (Programació I i Mètodes Numèrics) learning support Conversational Agent for the degrees of Computer Engineering and Mathematics at the Universitat de Barcelona, designed to assist the student with doubts about the teaching plan of the subjects and with general conceptual doubts. It also implements the delivery of exercises for students to solve, along with secure execution systems for C and Java languages through Docker. To facilitate the implementation of the mentioned exercises and their tests, a unit test library for the C language for teaching staff is presented, along with various examples. Some of the chosen examples are well-known numerical methods which have been expanded, extended, and theoretically formalized.

To enable the agent to perform the described functionalities, an architecture prepared for deployment via Docker-Compose is presented. This allows the agent to store and access specific data, update these in real time, execute corrections of the exercises, communicate with users through the Telegram interface, and keep records of the conversations per student for subsequent analysis.

Resum

Avui en dia, l'interès en els *xatbots* o bots de conversa ha augmentat dràsticament en conseqüència introducció dels Grans Models del Llenguatge (*LLM*) darrere de *ChatGPT* o *Google Bard*. Tot i les seves grans capacitats i altíssim nivell de comprensió i interacció en llenguatge natural, sofreixen de defectes com deliris i pèrdua del context de la conversa, fent-los poc segurs per entorns controlats o per la realització de tasques específiques.

Aquest projecte presenta un Agent Conversacional proveït per Rasa reforçat amb crides al model *GPT-3.5-turbo* d'OpenAI per acomplir tasques específiques que impliquen generació de llenguatge natural. El context del projecte és un Agent Conversacional de suport a l'aprenentatge multiassignatura (Programació I i Mètodes Numèrics) dels graus d'Enginyeria Informàtica i Matemàtiques de la Universitat de Barcelona, dissenyat per assistir a l'alumne amb dubtes sobre el pla docent de les assignatures que implementa i amb dubtes conceptuals generals. També implementa l'entrega d'exercicis a solucionar pels alumnes

juntament amb sistemes d'execució segura dels llenguatges C i Java mitjançant Docker per proporcionar les correccions. Per facilitar la implementació dels exercicis esmentats i els seus tests, es presenta una llibreria de tests unitaris de llenguatge C per al personal docent juntament amb diversos exemples. Alguns dels exemples escollits són mètodes numèrics coneguts els quals s'han ampliat, expandit i formalitzat teòricament.

Per l'agent poder realitzar les funcionalitats descrites, es presenta una arquitectura preparada pel desplegament mitjançant Docker-Compose que permet a l'agent emmagatzemar i accedir a dades especifiques, l'actualització d'aquestes en temps real, l'execució de les correccions dels exercicis, la comunicació amb les usuàries a través de la interfície Telegram i registres de les converses agent-alumne pel seu posterior anàlisi.

Abstract (Castellano)

Hoy en día, el interés en los chatbots ha aumentado drásticamente debido a la introducción de los Enormes Modelos del Lenguaje (LLM) detrás de *ChatGPT* o *Google Bard*. Pese a sus grandes capacidades y altísimo nivel de comprensión e interacción en lenguaje natural, estos sufren de defectos, como delirios y pérdida del contexto de la conversación, volviéndolos poco seguros en entornos controlados o tareas específicas.

Este proyecto presenta un Agente Conversacional provisto por Rasa, respaldado con llamadas al modelo *GPT-3.5-turbo* de OpenAI para realizar tareas específicas que implican generación de lenguaje natural. El contexto del proyecto es un Agente Conversacional de apoyo al aprendizaje multiasignatura (Programación I i Métodos Numéricos) de los grados de Ingeniería Informática y Matemáticas de la Universitat de Barcelona, diseñado para asistir al estudiante resolviendo dudas sobre el plan docente de las asignaturas que implementa y sobre dudas conceptuales generales. También implementa la entrega de ejercicios a solucionar por los alumnos, juntamente con sistemas de ejecución segura de los lenguajes C y Java mediante Docker. Para facilitar la implementación de dichos ejercicios y sus tests, se presenta una librería de tests unitarios del lenguaje C para el personal docente, juntamente con múltiples ejemplos. Algunos de dichos ejemplos son métodos numéricos conocidos, los cuales se han ampliado, expandido y formalizado teóricamente.

Para el agente poder realizar las funcionalidades descritas, se presenta una arquitectura preparada para el *deployment* mediante Docker-Compose que permite a l'agente almacenar y acceder a datos específicos, la actualización de estos en tiempo real, la ejecución de las correcciones de los ejercicios, la comunicación con las usuarias mediante la interficie Telegram y registros de las conversaciones agente-alumne para su posterior análisis.

Agraïments

Primerament, vull agrair a les meves dues tutores, la Inma i la Susana, la paciència i gentilesa en dirigir tant aquest treball com el tutoritzat que l'ha escrit. Sense elles, les reunions setmanals i la seva gran docència no me n'hagués sortit.

Vull agrair també a la meva família, mare i pare i (en un lloc especial) avi, haver-me iniciat i aguantat en el camí que m'ha dut fins aquí, a tancar aquesta etapa, i a tot el suport que he rebut per part seva. També, sense ells, tampoc ho hauria aconseguit.

Igualment vull agrair a l'Héctor Alarcón i a la Marta Martí les estones de treballar plegats i de simbiosi mútua en els moments més baixos d'aquest procés. Sense ells, probablement no hauria reimplementat el projecte de zero, i aquest treball seria molt menys del que és.

Sens dubte no li deixaré d'agrair mai a la meva parella, la Mariona, haver tingut el privilegi de compartir el fer aquest escrit a la vegada que ella feia el seu, amb tot l'amor i suport que m'ha donat per poder-lo acabar, només per continuar compartint tot el que ara vindrà.

I finalment, m'agradaria fer un agraïment i, sobretot, dedicar aquest treball a en Lluís Garrido, professor de la facultat que malauradament ens va ser arravatat abans d'hora. Ell, amb tota la seva tendresa i altruisme, em va donar l'oportunitat de fer aquest treball, i jo l'he fet el millor que he sabut. Així que, per això, li dedico al meu amic i professor Lluís, a la memòria de la bondadosa persona que sempre ha estat.

Índex

Index de figures

Capítol 1

Introducció

Aquest capítol introdueix el projecte, començant pel context i la motivació, continuant amb els objectius i la planificació dels mateixos, i acabant amb la descripció del contingut de la memòria.

1.1 Context i Motivació

El concepte d'Agent Conversacional és tan antic com els fonaments de la computació moderna. Les primeres mencions d'aquest són amb Alan Turing i el test que porta el seu nom, on es volia comprovar que una màquina (l'Agent Conversacional) pogués simular el comportament humà. A mesura que la tecnologia ha anat avançant, s'han desenvolupat diversos agents conversacionals amb les tecnologies del moment, des de la primera ELIZA [23] detectant certes paraules o patrons a les frases de les usuàries fins als refinadíssims models de llenguatge grans [24](o *LLM: Large Language Model*) com GPT-4 o Google Bard.

L'estudi i desenvolupament d'Agents Conversacionals es classifica actualment en una branca de la intel·ligència artificial anomenada Processament del Llenguatge Natural (o *NLP: Natural Language Processing*) que en sí mateixa es pot partir en dues parts, la Comprensió del Llenguatge Natural (o NLU: *Natural Language Undertanding)* i Generació del Llenguatge Natural (o *NLG: Natural Language Generation*). Tot i que els models d'Agents Conversacionals més recents implementin ambdues capacitats (*NLG* i *NLU*) no és estrany trobar implementacions d'Agents amb *NLU* que responen amb missatges predefinits.

Avui en dia es poden classificar els Agents Conversacionals en dos grans tipus. El primer serien els Agents Conversacionals Generals, sistemes que han estat dissenyats per a comprendre i respondre a una àmplia varietat de preguntes i sol·licituds dels usuaris sense importar-ne el domini; els representants d'aquesta categoria utilitzen tècniques avançades

de *NLP* com l'aprenentatge automàtic (o *machine learning*) juntament amb una enorme quantitat de dades per entrenar els models. El segon tipus serien els Agents Conversacionals Específics, els quals són dissenyats per dominis concrets, tenen un coneixement del món més limitat i estan pensats per assistir en una sèrie de tasques definides prèviament.

L'objectiu dels Agents Conversacionals Específics és oferir una experiència d'usuari més natural, assimilant-la a una interacció humana per a fer-la més amena i senzilla per l'usuari. Aquests Agents més concrets han proliferat en sistemes de salut, atenció al client [11] i assistents virtuals per dubtes a pàgines web. Històricament, les tasques delegades a aquests Agents han estat estretament lligades a les capacitats de la tecnologia que les proveïa; els primers Agents Conversacionals responien a l'usuari detectant patrons al seu missatge o a paraules concretes, que el feien escollir entre diverses respostes predefinides o navegar un menú amb la interfície d'un xat. En canvi, quan l'aprenentatge automàtic (o *Machine Learning*) demostra ser capaç de proporcionar els Agents la capacitat de mantenir converses més complexes amb els usuaris mitjançant llenguatge natural, les tasques poden esdevenir més complexes, i la detecció de patrons passa a implementar-se mitjançant aprenentatge automàtic, fent l'experiència d'usuari molt més àgil i satisfactòria.

En l'àmbit educatiu i de suport a l'aprenentatge, els Agents Conversacionals han estat usats com a assistents: recordant a les alumnes certes tasques, ajudant-les a organitzar-se o responent dubtes simples[11]. Amb l'arribada de l'aprenentatge automàtic s'aconsegueix proveir els Agents amb moltes altres capacitats, com la personalització de continguts per a cada usuari o recordant certes preferències individuals.

Aquest projecte es situa en la mateixa línia de treball de projectes anteriors: '*Agentes conversacionales educativos*' de Fernández Fernández[6] i '*Dockers para un sistema de corrección de ejercicios en un chatbot*' de Doménech Puig[5]. En aquests projectes s'implementava l'Agent LovelaceUB. Via Telegram, l'usuari podia preguntar dubtes, demanar exercicis, i enviar la solució a l'agent perquè els corregís mitjançant un Docker. L'Agent, dissenyat per l'assignatura de Programació I del grau d'Informàtica, emprava la tecnologia AIML (*Artificial Intelligence Modelling Language*) capaç de detectar patrons en els missatges de l'usuari i escollir-ne una resposta predefinida, fet que limitava molt la interacció Agent-Usuari. També, el disseny de l'arquitectura de l'aplicació i la seva implementació impedien afegir noves assignatures fàcilment i, per tant, també nous correctors per altres llenguatges de programació de manera senzilla, afectant l'escalabilitat del projecte.

En aquest context, el projecte actual proposa un Agent Conversacional proveït per Rasa, un entorn que empra aprenentatge automàtic per la identificació de la intenció darrera dels missatges de l'usuari i decidir-ne la millor acció a realitzar com a resposta. Per la generació de lleguatge natural, a certes intencions de l'usuari es respon amb un missatge generat pel model *GPT-3.5-turbo* d'OpenAI[27]. A nivell d'arquitectura es proposa la implementació d'una aplicació preparada pel seu desplegament mitjançant Docker, permetent-ne l'ampliabilitat en el trànsit de les dades i en afegir nous correctors de diferents llenguatges de programació. Per emmagatzemar els exercicis, es proposa posar-los en repositoris de

GitHub i implementar-ne l'actualització quan es detectin canvis, addicions o supressions de documents.

El nou Agent incorpora les següents funcionalitats bàsiques dels dos projectes anteriors: demanar exercicis i corregir-los, contestar dubtes sobre el pla docent i sobre el temari de l'assignatura. Addicionalment, s'hi ha afegit l'assignatura Mètodes Numèrics 1 del grau de Matemàtiques juntament amb un corrector del llenguatge C, llenguatge emprat en l'assignatura. Per a fer-ho, s'ha aprofundit i formalitzat el Mètode de la Potència i el problema d'Interpolació Polinomial amb el mètode de Newton, amb l'objectiu de crear enunciats i tests unitaris que servissin d'exemple per futurs exercicis i tests. Els mètodes han estat escollits pel seu ús d'estructures de dades concretes, com matrius i vectors, pels quals es dona una llibreria de correcció amb funcions facilitant la comparació de les estructures i la seva gestió de memòria, que permetrà facilitat la implementació d'enunciats i tests per futurs exercicis. Pel corrector de Java s'han implementat els tests unitaris dels exercicis que havien preparat els altres dos projectes.

En aquest projecte s'han emprat i/o ampliat coneixement de les assignatures Sistemes Operatius I, Sistemes Operatius II, Software Distribuït, Enginyeria del Software, Disseny de Software, Bases de Dades, Factors Humans i Computació, Intel·ligència Artificial, Mètodes Numèrics I i Mètodes Numèrics II.

1.2 Objectius

Els objectius del projecte eren estudiar l'ampliació de l'agent antic, tant en correctors d'exercicis com en conversa. Pels correctors, es volia revisar la implementació del corrector de Java i ampliar el bot amb un corrector per C de les assignatures de Matemàtiques. En la conversa, es volia canviar la tecnologia d'*NLU* (*Natural Language Understanding*) que contenia el bot, basada en reconeixement de patrons, per una que emprés d'aprenentatge automàtic. Per acabar, es volia desplegar l'aplicació a un servei d'allotjament per deixar-la preparada pel seu ús. Finalment, no s'ha reutilitzat quasi res del codi dels projectes originals (per raons explicades més endavant al document) i, per tant, els objectius canviaren a la implementació d'una nova arquitectura completa amb sistemes de correccions pel bot de conversa i el posterior disseny i implementació de l'agent conversacional Lovelace, proveït pel marc conversacional Rasa, juntament amb els mètodes numèrics i la llibreria de tests unitaris de C.

1.2.1 Objectius Generals

Per tant, els objectius generals es dividiren en 3 fases:

1. Revisió i Desplegament: Revisar, compendre i desplegar els dos projectes anteriors.

2. Disseny i implementació de la nova arquitectura: dissenyar una aplicació adient a les capacitats que volíem atorgar a la Lovelace amb una assignatura nova (Mètodes Numèrics) i correctors dels llenguatges C i Java

3. Disseny i desenvolupament de l'Agent en Rasa: Un cop feta l'arquitectura, queda desenvolupar la Lovelace i les capacitats que tindrà.

4. Desplegament a un servei d'allotjament.

1.2.2 Objectius Específics

S'han subdividit en els següents subobjectius:

- Revisió i Desplegament:
  - Execució i comprensió del codi dels anteriors projectes.
  - Revisió i canvi d'arquitectura pel desplegament.
  - Desplegament.
- Disseny i implementació de l'aplicació:
  - Recerca de llibreries asíncrones de Python amb tractament de Bots de Telegram i l'apropiada gestió d'esdeveniments a través de la xarxa.
  - Recerca i aprenentatge de Docker, Docker-Compose i Docker SDK per Python.
  - Recerca de llibreries de gestió de GitHub dins de Python. Implementació de funcions de gestió de repositoris locals.
  - Disseny de la base de dades i les abstraccions d'interacció amb ella mitjançant Python
  - Implementació de l'actualització dels repositoris dels exercicis de les assignatures.
  - Implementació de la comunicació amb Telegram.
  - Implementació de les comandes de Telegram per interactuar amb l'Agent.
  - Recerca i implementació de *hashing* i encriptació pel correu i la id dels usuaris.
  - Implementació del Corrector de Java
  - Disseny de la llibreria de Correcció per Mètodes Numèrics.
  - Desenvolupament de les solucions, els tests i els enunciats del Mètode de la Potència, Diferències Dividides.
  - Formalització matemàtica dels mètodes mencionats.

  - Implementació d'un nou Corrector del llenguatge C
  - Implementació d'un patró *Factory* per la creació dels contenidors de Docker.
  - Disseny i implementació de la comunicació dels resultats dels tests: procés dels resultats i enviament entre mòduls.
  - Testatge profund de les capacitats i resultats del corrector.
  - Uniformització de la comunicació intermodular i dels errors relacionats amb l'accés a la base de dades.
  - Aixecament simultani amb Docker-Compose: gestió de xarxes virtuals, *healthchecks* i imatges predefinides.
- Disseny i desenvolupament de l'Agent proveït per Rasa
  - Recerca de l'entorn Rasa i les seves capacitats.
  - Disseny de les funcionalitats de l'agent.
  - Desenvolupament, testatge i implementació de les capacitats de l'agent.
  - Connexió de l'Agent amb *GPT-3.5-turbo*.
  - Acoblar l'arquitectura de Rasa a l'aplicació.
  - Coordinar l'estat de l'entorn Rasa amb l'entorn Telegram.
- Desplegament de l'aplicació

1.3 Planificació

Els objectius plantejats s'han planificat en les següents fases i amb la temporalització que s'indica a continuació:

- Fase 1 (13-02-23 - 23-02-23): Comprensió de l'aplicació anterior i desplegament d'aquesta.
- Fase 2 (23-02-23 - 15-04-23): Disseny i implementació de l'aplicació base de l'agent conversacional, tests i llibreria matemàtica.
- Fase 3 (15-04-23 - 15-05-23): Disseny i desenvolupament de l'agent conversacional de Rasa.
- Fase 4 (16-05-23 - 05-06-23): Connexió Rasa-Telegram i finalització del codi del proejcte.
- Fase 5 (05-06-23 - 13-06-23): Acabar i revisar la memòria del projecte.

1.4 Estructura de la Memòria

Aquesta memòria presenta els següents capítols:

- Capítol 1. Introducció al projecte, objectius i motivació.
- Capítol 2. Formalització matemàtica sobre el Mètode de la Potència i les Diferències Dividides
- Capítol 3. Explicació del funcionament de les tecnologies més importants emprades en aquest projecte.
- Capítol 4. Contextualització dels treballs anteriors relacionats amb el projecte LovelaceUB
- Capítol 5. Explicació del marc conversacional Rasa pel disseny i implementació d'agents conversacionals.
- Capítol 6. Anàlisi de les funcionalitats de l'Agent i la interacció de l'usuari amb ell mitjançant casos d'ús.
- Capítol 7. Disseny de l'aplicació i els seus submòduls.
- Capítol 8. Implementació de l'aplicació mitjançant Docker i detalls tècnics.
- Capítol 9. Anàlisi dels objectius aconseguits al llarg del projecte.
- Capítol 10. Elements del projecte amb marge de millora i futur del desenvolupament de l'aplicació.
- Annex: Manual tècnic per l'ús del projecte.

1.5 Nomenclatura Lingüística

En aquest escrit s'ha intentat fer el màxim ús de la terminologia Catalana a la vegada que intentant evitar activament l'Anglesa, tot i que molts termes malauradament no s'ha pogut evitar emprar l'Anglicisme equivalent o directament la paraula Anglesa. Tot i això, per la naturalesa del treball, s'ha hagut de fer una petita investigació sobre com referir-se al terme *chatbot* en Català.

L'anglicisme més comú per referir-se a Agents Conversacionals és *xatbot*, traduït directament de la llengua anglosaxona i no acceptat pel Diccionari de l'Institut d'Estudis Catalans (DIEC). Per evitar usar aquest terme i seguint les directrius de la Universitat

de Barcelona[21], la Universitat Oberta de Catalunya[22] i el mateix TERMCAT[20], s'ha d'emprar el terme 'bot' (més general i, per tant, depenent del context) o 'bot de conversa'.

En aquest treball ens hi adrecem també com a 'Agent Conversacional' o 'Bot Conversacional', termes que tot i no estar reconeguts directament, empren vocabulari català. La decisió rau en el fet que és més senzill escriure mots composts com els dos anomenats al principi del paràgraf que no pas una denominació sintagmàtica com 'bot de conversa' tot i ser estrictament més correcta. Així i tot, per donar varietat a certes parts del text o quan ho hem vist necessari de manera contextual, s'ha emprat el mot *xatbot*, tot i que molt esporàdicament.

Seguidament, al ser el nom de l'agent Lovelace de gènere femení, el terme 'bot', 'bot de conversa' o 'agent conversacional' s'han adreçat en femení. També, per evitar el masculí genèric per defecte, s'ha intentat emprar emprar mots en gènere neutre per referir-se a col·lectius, com 'alumne', ja que és una paruala invariant del gènere. En altres llocs del text, s'ha pres la llibertat d'emprar el femeni genèric (concretament, en el terme usuàries).

Per acabar i amb la claredat en ment, es troba una equivalència a continuació traduïdes del Català a l'Anglès de les quals no hem trobat una traducció estandarditzada però ens hem pres la llibertat de traduir:

- Desplegar una aplicació: *Deploy an application*
- El Marc Conversacional de Rasa: *Rasa's conversational framework*
- Atacar una adreça: *Atack an address*
- Construir el contenidor: *Build the container*
- Aquest *xatbot* està proveït per Rasa: *This chatbot is empowered by Rasa.* Particularmanet no hi ha una expressió que traudeïxi el significat de *empowered by* directament, però l'autor es pren la llibertat de considerar aquesta aproximació com a sinònim.
- *Dockeritzar* com a verb del substantiu *Docker* és usat (i marcat en cursiva) al llarg de tot l'escrit per evitar la traducció fidedigne següent: 'S'han *dockeritzat* tots els mòduls d'aquesta aplicació' passaria a ser 'S'ha construït una imatge i contenidors de Docker per a tots els mòduls d'aquesta aplicació'. És a dir *dockeritzar* guanya el significat de la contrucció de la imatge o de la imatge i el contenidor.

Capítol 2

Formalització

En aquest capítol s'han formalitzat els dos mètodes numèrics dels que s'han implementat les respectives solucions, tests unitaris en C i els enunciats. La secció de Diferències Dividides ha estat adaptada de *'Numerical Mathematics'*[1] i de *'Introduction to Numerical Computation'*[12]. La secció del Mètode de la Potència també s'ha emprat *'Numerical Mathematics'*[1] juntament amb *'An Introduction to Numerical Analysis'*[2] i *'Numerical Analysis'*[16].

2.1 Diferències Dividides

El mètode de diferències dividides és un mètode d'interpolació per trobar la solució al problema de Lagrange emprant la anomenada forma de Newton del polinomi interpolador.

2.1.1 Preliminars

Definició 2.1.1. ***Polinomi Interpolador:*** *Sigui f una funció qualsevol i $x_0, \ldots, x_n$ punts qualssevol del domini d'aquesta. Diem que la funció P interpola a f si $P(x_i) = f_i, i = 0, ..., n$, és a dir, si les funcions f i P coincideixen als punts x_i, que anomenarem nodes. Concretament, quan P sigui un polinomi, direm que és el polinomi interpolador d'f de grau n*

Teorema 2.1.2. ***Existència i Unicitat del polinomi interpolador****: Siguin $x_0, ..., x_n$ nodes arbitraris i diferents entre ells. Per a valors arbitraris $f_0, ..., f_n$, existeix un únic polinomi P amb $gr(P) \leq n$ que compleix la definició d'interpolació, és a dir $P(x_i) = f_i, i = 0, ..., n$*

Demostració. Per provar el teorema s'ha de construir el polinomi avaluant a tots els

nodes. Consideris P com el polinomi interpolador amb grau menor o igual a n en la base de polinomis naturals $P(x) = a_0 + a_1x + \ldots + a_nx^n$, sent l'expressió següent l'avaluació als $n+1$ nodes:

$$i = 0 : P(x_0) = a_0 + a_1x_0 + \ldots + a_nx_0^n = y_0$$

$$\ldots$$

$$i = n : P(x_n) = a_0 + a_1x_n + \ldots + a_nx_n^n = y_n$$

El sistema anterior es pot escriure en forma matricial:

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}$$

Denotis aquest sistema com $V_na = y$, on V_n és la matriu de Vandermonde. Tenint ja el sistema, es vol veure que el determinant de la matriu de Vandermonde no s'anul·la per provar que el sistema que forma és compatible determinat. Usant la invariància del determinant per combinacions lineals de columnes i restant a cada columna la columna anterior multiplicada per x_0, obtenim el següent determinant:

$$D_n(x_0, \ldots, x_n) = \begin{vmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & x_1 - x_0 & x_1(x_1 - x_0) & x_1^2(x_1 - x_0) & \cdots & x_1^{n-1}(x_1 - x_0) \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n - x_0 & x_n(x_n - x_0) & x_n(x_n - x_0) & \cdots & x_n^{n-1}(x_n - x_0) \end{vmatrix}$$

i anomenant a la submatriu següent B,

$$B = \begin{bmatrix} x_1 - x_0 & x_1(x_1 - x_0) & x_1^2(x_1 - x_0) & \cdots & x_1^{n-1}(x_1 - x_0) \\ x_2 - x_0 & x_2(x_2 - x_0) & x_2^2(x_2 - x_0) & \cdots & x_2^{n-1}(x_2 - x_0) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_n - x_0 & x_n(x_n - x_0) & x_n^2(x_n - x_0) & \cdots & x_n^{n-1}(x_n - x_0) \end{bmatrix}$$

és compleix que $D_n(x_0, \ldots, x_n) = det(B)$, i com que el factor $x_{i+1} - x_0$ és comú a la columna i-èssima, es poden treure els factors per obtenir la recurrència següent:

$$det(B) = (x_1 - x_0)\ldots(x_n - x_0) \begin{vmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^{n-1} \end{vmatrix} = \prod_{1 \leq i \leq n} (x_i - x_0) D_{n-1}(x_1, \ldots, x_n)$$

on $V_{n-1}(x_1, \ldots, x_n)$ és la matriu de Vandermonde amb un node i un grau menys que l'original. Així es pot expressar fàcilment l'expressió del determinant fent la recurrència n vegades com:

$$det(V_n(x_0, \ldots, x_n)) = \prod_{0 \leq j < i \leq n} (x_i - x_j)$$

Com que aquest determinant és clarament diferent de zero, vol dir que el sistema original és determinat i té solució única, sent aquesta el polinomi interpolador, provant ergo l'existència i unicitat del polinomi interpolador.

□

Ara que ja s'ha vist que el polinomi existeix i és únic, es defineix el polinomi interpolador de Lagrange (o la forma de Lagrange) per abordar el problema en qüestió:

Definició 2.1.3. ***Forma de Lagrange****: Es diu que el polinomi interpolador està en forma de Lagrange si s'escriu com:*

$$P(x) = \sum_{i=0}^{n} f(x_i) L_i(x) \text{ amb } L_i(x) = \prod_{j=0 j \neq i}^{n} \frac{x - x_j}{x_i - x_j}$$

Això és consistent i possible ja que $\{L_0, \ldots, L_n\}$ són base de polinomis i en destaca que avaluada als nodes funciona com una delta de Kronecker[1], és a dir:

$$L_i(x_j) = \delta_{i,j} = \begin{cases} 0 & \text{si } i \neq j \\ 1 & \text{si } i = j \end{cases}$$

que ens assegura que compleix la definició de polinomi interpolador.

Per acabar, l'error d'interpolació entre el polinomi P i la funció f es descriu amb el següent teorema:

Teorema 2.1.4. ***Error del Polinomi Interpolador de Lagrange:*** *Sigui $f \in \mathcal{C}^{n+1}(I)$ on I és un interval contenint tots els nodes $x_0, ..., x_n$. Considerem P com el polinomi interpolador de grau menor o igual que n, es compleix:*

$$f(x) - P(x) = \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x - x_0) \ldots (x - x_n)$$

per algun $\xi(x) \in I$

[1]Una funció que s'anul·la si els dos index son diferents, i no ho fa si són iguals

Demostració. S'usarà el Teorema de Rolle: si $g \in \mathcal{C}([a,b]), g \in \mathcal{C}^1(]a,b[)$ i $g(a) = g(b)$, llavors $\exists \eta \in]a,b[$: $g'(\eta) = 0$. Escollint un punt arbitrari $\hat{x} \in]a,b[$ i escrivim l'error $f(\hat{x}) - P(\hat{x})$ amb la forma

$$f(\hat{x}) - P(\hat{x}) = A(\hat{x} - x_0) \dots (\hat{x} - x_n)$$

Si $\hat{x} = x_i, i \in \{0, \dots, n\}$, s'ha demostrat el teorema, ja que

$$f(\hat{x}) - P(\hat{x}) = 0 - 0 = A(x_i - x_0) \dots (x_i - x_i) \dots (x_i - x_n) = 0$$

que també fa donar zero ambdós valors de la igualtat enunciats.

Assumeixis doncs que $\hat{x} \neq x_i, i = 0, \dots, n$. Es vol trobar el valor de la constant A, que serà la que el teorema prediu. Per fer-ho, consideris l'expressió anterior com a funció auxiliar:

$$\psi(x) = f(\hat{x}) - P(\hat{x}) - A(\hat{x} - x_0) \dots (\hat{x} - x_n)$$

Aquesta funció té $n + 2$ arrels:

- $\psi(\hat{x}) = 0$ per com la hem definit
- Els $n + 1$ nodes: $\psi(x_i) = f(x_i) - P(x_i) - A \cdot 0 = 0, i = 0, \dots, n$

Per les hipòtesis, ψ és contínuament diferenciable $n + 1$, i pel teorema de Rolle es veu que ψ' té almenys un zero en cada subinterval entre les dues arrels de ψ, per tant, ψ' té almenys $n + 1$ arrels diferents. Anàlogament, entre dos arrels de ψ' hi ha una arrel de ψ'', per tant ψ'' té n arrels diferents. Repetint aquesta recurrència, es veu que la derivada $n + 1$-èssima de ψ ha de tenir un zero a l'interval format per $\hat{x}, x_0, \dots, x_n$:

$$\psi^{(n+1)}(\xi) = 0$$

i derivant s'obté l'expressió:

$$\psi^{(n+1)}(x) = f^{(n+1)}(x) - A \cdot (n + 1)!$$

Aïllant A s'obté l'expressió mencionada al teorema, on ξ depèn de $\hat{x}$, però com que $\hat{x}$ és un punt arbitrari, el teorema queda provat. □

2.1.2 Fórmula d'Interpolació de Newton

Tot i el polinomi existir i ser únic per funció, es pot expressar en diferents formes, comportant diferents avantatges i inconvenients. Per les diferències dividides s'usa la forma de Newton.

Definició 2.1.5. ***Forma de Newton del Polinomi de Lagrange:*** *Sigui f una funció i P interpolant f a $x_0, \ldots, x_n$. La forma de Newton de P és:*

$$P(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \ldots + a_n(x - x_0)\ldots(x - x_{n-1})$$

Aquesta també es pot expressar de la manera següent:

$$\begin{aligned}
\varphi_0(x) &= 1 \\
\varphi_1(x) &= (x - x_0) \\
&\ldots \\
\varphi_n(x) &= \prod_{i=0}^{n-1}(x - x_i)
\end{aligned}$$

i per tant:

$$P(x) = \sum_{i=0}^{n} \alpha_i \varphi_i(x)$$

Aquesta forma del polinomi és consistent per un argument semblant al de la forma de Lagrange: els polinomis $\{\varphi_0, \ldots, \varphi_n\}$ són base de polinomis, per tant es pot expressar-ne qualsevol en funció d'aquesta. Expressar un polinomi en base de Newton comporta dos avantatges conforme fer-ho en base natural $\{1, x, x^2, \ldots, x^n\}$:

1. Cost computacional reduït en avaluació: Emprant la regla de Horner escrivint el polinomi de manera telescòpica:

   $$P(x) = a_0 + (x - x_0)[a_1 + (x - x_1)[a_2 + (x - x_2)[\ldots]]]$$

   avaluar-lo costa $2n$ operacions, (un producte i una suma per cada terme del polinomi) mentre que en base natural son moltes més.

2. Augmentar el grau: Com que la forma és recursiva, per afegir un grau més només s'ha de col·locar el terme extra, evitant haver de recalcular tots els coeficients anteriors.

Específicament, els coeficients del polinomi de Newton compleixen una propietat especial, que és la que dona nom a les diferències dividides:

Proposició 2.1.6. ***Propietat de les Diferències Dividides:*** *Sigui el polinomi interpolador d'f en forma de Newton: $P(x) = \sum_{i=0}^{n} a_i \varphi_i(x)$. Anomenant els coeficients de la següent manera:*

$$a_0 = f[x_0], a_1 = f[x_0, x_1], \ldots, a_k = f[x_0, \ldots, x_k]$$

Podem obtenir tots els coeficients del polinomi seguint aquesta regla:

$$a_0 = f[x_0] = f(x_0)$$

$$a_1 = f[x_0, x_1] = \frac{f[x_0] - f[x_1]}{x_1 - x_0}$$

$$\vdots$$

$$a_k = f[x_0, \ldots, x_k] = \frac{f[x_1, \ldots, x_k] - f[x_0, \ldots, x_{k-1}]}{x_k - x_0}$$

Anomenem a $f[x_0, \ldots, x_k]$ la diferència dividida k-èssima.

Per la demostració es necessita el lema (2.1.7) que es troba just després de la demostració.

Demostració. Es provarà la formula de l'enunciat buscant el coeficient principal de $Q(x)$, a_k, fent-ho per inducció. El cas base és trivial, així que anem directament al cas inductiu. Siguin $R(x)$ el polinomi interpolador de f als nodes $x_1, \ldots, x_k$ i $S(x)$ el polinomi interpolador de f als nodes $x_0, \ldots x_{k-1}$, i considerem les respectives representacions en base natural:

$$R(x) = b_0 + b_1 x + \ldots + b_{k-1} x^{k-1}$$
$$S(x) = c_0 + c_1 x + \ldots + c_{k-1} x^{k-1}$$

Usant el lema (2.1.7), la construcció del següent polinomi:

$$Q(x) := R(x) + \frac{x - x_k}{x_k - x_0}(R(x) - S(x))$$

$Q(x)$ interpolarà a f en $x_0, \ldots, x_k$, i a més a més, per la hipòtesi inductiva

$$b_{k-1} = f[x_1, \ldots, x_k],\ c_{k-1} = f[x_0, \ldots, x_{k-1}]$$

És clar que el coeficient de terme de grau més alt de $Q(x)$ per definició clarament és:

$$\frac{1}{x_k - x_0}(b_{k-1} - c_{k-1})$$

I com s'ha vist ja, transpira que:

$$a_k = \frac{1}{x_k - x_0}(b_{k-1} - c_{k-1}) = \frac{f[x_1, \ldots, x_k] - f[x_0, \ldots, x_{k-1}]}{x_k - x_0} = f[x_0, \ldots, x_k]$$

demostrant així la proposició. □

Lema 2.1.7. *Siguin $R(x)$ el polinomi interpolador de f als nodes $x_1, \ldots, x_k$ i $S(x)$ el polinomi interpolador de f als nodes $x_0, \ldots x_{k-1}$. Llavors,*

$$Q(x) := R(x) + \frac{x - x_k}{x_k - x_0}(R(x) - S(x))$$

és el polinomi interpolador de f a $x_0, \ldots, x_k$.

Demostració. Es vol veure que $Q(x)$ interpola a f en els nodes $x_0, \ldots, x_n$, per tant hem de veure que $Q(x_i) = f_i$ per $i = 0, \ldots n$. Mirem tots els casos de i i veiem que compleix:

- $i = 0$:

$$Q(x_0) = R(x_0) + \frac{x_0 - x_k}{x_k - x_0}(R(x_0) - S(x_0)) = R(x_0) - (R(x_0) - S(x_0)) = S(x_0) = f(x_0)$$

- $i = k$:

$$Q(x_k) = R(x_k) + \frac{x_k - x_k}{x_k - x_0}(R(x_k) - S(x_k)) = R(x_k) = f(x_k)$$

- $i = 1, \ldots, k-1$:

$$Q(x_i) = R(x_i) + \frac{x_i - x_k}{x_k - x_0}(R(x_i) - S(x_i)) = R(x_i) = f(x_i)$$

On s'ha usat que $R(x_i) - S(x_i) = 0$ per la condició d'interpol·lació en els tres casos. □

Ara ja podem procedir a la demostració de la proposició de les diferències dividides:

2.1.3 Estabilitat i Error del Polinomi Interpolador

Per parlar de l'estabilitat del polinomi interpolador, s'ha de definir en primer lloc el nombre de condició d'un mètode numèric i, posteriorment, com trobar el nombre de condició del problema d'interpolació de Lagrange.

Consideris el següent problema: troba la x tal que

$$F(x, d) = 0$$

on d són els conjunts de dades dels quals en depèn la solució i F és la relació entre x i d. Es diu que el problema F és *estable* si admet només una única solució x que depèn amb continuïtat de les dades. En canvi, es diu que el problema és *inestable* si no compleix la propietat d'estabilitat i que, per tant, abans d'aplicar-li un mètode numèric ha de ser tractat.

La propietat de que x sigui dependent de manera contínua de les dades significa que petites pertorbacions a aquestes comporten petits canvis a la solució x. Concretament, si δd és una petita pertorbació a les dades i δx el canvi conseqüent a la solució de manera que:

$$F(x + \delta x, d + \delta d) = 0 \tag{2.1.1}$$

i per tant se'n dedueix

$$\forall \eta,\, \exists K(\eta, d) :\, \|\delta d\| < \nu \Rightarrow \|x\| \leq C(\eta, d)\|\delta d\|$$

on $C(\eta, d)$ és el nombre de condició.

Definició 2.1.8. ***Nombre de Condició:*** *Donat un problema $F(d, x) = 0$, es defineix el nombre de condició absolut com*

$$C_{abs}(d) = \sup_{\delta d \in D} \frac{\|\delta x\|}{\|\delta d\|}$$

i, si $d \neq 0$ i $x \neq 0$, el nombre de condició relatiu com

$$C_{rel}(d) = \sup_{\delta d \in D} \frac{\|\delta x\|/\|x\|}{\|\delta d\|/\|d\|}$$

on D és un entorn de l'origen. Concretament, D és el conjunt de pertorbacions admissibles en el context del problema, és a dir, les pertorbacions que no deformen el problema descrit a (2.1.1).

Els conceptes d'estable i inestable es poden redefinir segons el nombre de condició. Donat un problema concret, direm que el problema és *estable* sí té un $C(d)$ petit per a qualsevol d i *inestable* si té un nombre de condició $C(d)$ gran. Cal matisar que els adjectius 'gran' i 'petit' depenen del problema en qüestió.

A continuació es dedueix el nombre de condició del polinomi interpolador seguint els passos descrits fins ara. Siguin $f(x)$ i $P(x)$ el seu polinomi interpolador on per comoditat s'escriurà $f(x_i) = y_i$. Consideris les parelles de dades $(x_i, y_i), i = 0, \ldots, n$ dels nodes d'interpolació i les imatges d'aquests. Ara, denotis $\tilde{y}_i$ com una pertorbació de les dades d'y[2].

El polinomi pertorbat $\tilde{P}(x)$ és el que s'obté amb la definició del polinomi interpolador i les dades pertorbades:

$$P(x) = \sum_{i=0}^{n} y_i L_i(x),\ \tilde{P}(x) = \sum_{i=0}^{n} \tilde{y}_i L_i(x)$$

[2]Consideris una pertorbació quan y_i i $\tilde{y}$ compleixen $|y_i - \tilde{y}_i| < \varepsilon$, on ε és suficientment petita.

Cal notar que el polinomi pertorbat $\tilde{P}$ també interpola la funció f, però és diferent. Llavors, com de diferents són $\tilde{P}$ i P?

$$|P(x) - \tilde{P}(x)| = \left| \sum_{i=0}^{n} (y_i - \tilde{y}_i) L_i(x) \right| \leq \max_{i=0,\dots,n} |y_i - \tilde{y}_i| \sum_{i=0}^{n} |L_i(x)| \tag{2.1.2}$$

Recordem al lector la definició de la norma del màxim, ja que s'emprarà en breus.

Definició 2.1.9. ***Norma del Màxim:*** *Sigui $f \in \mathcal{C}^0([a,b])$. En definim la seva norma del màxim com*

$$\|f\|_\infty = \max_{x \in [a,b]} |f(x)| \tag{2.1.3}$$

Ara, prenent la norma del màxim a l'expressió (2.1.2) anterior obtenim l'expressió següent, que depenen de les imatges (del polinomi original i el pertorbat) i de la base de Lagrange.

$$\|P(x) - \tilde{P}(x)\|_\infty = \max_{x \in [a,b]} |P(x) - \tilde{P}(x)| \leq \max_{i=0,\dots,n} |y_i - \tilde{y}_i| \max_{x \in [a,b]} \sum_{i=0}^{n} |L_i(x)| \tag{2.1.4}$$

Definició 2.1.10. ***Constant de Lebesgue:*** *Definim la constant de Lebesgue del polinomi interpolador de grau n com*

$$\Lambda_n = \max_{x \in [a,b]} \sum_{i=0}^{n} |L_i(x)|$$

Per tant, seguint (2.1.4) tenim:

$$\|P(x) - \tilde{P}(x)\|_\infty \leq \Lambda_n \max_{i=0,\dots,n} |y_i - \tilde{y}_i|$$

De l'expressió transpira que efectivament la constant de Lebesgue és el nombre de condició del problema de la interpolació de Lagrange. Per tant, petites pertorbacions a les dades d'entrada portaran a petits canvis al polinomi interpolador si i només si la constant de Lebesgue és petita, fent el problema estable. En canvi, si és gran, magnificaran molt la pertorbació fent el problema inestable. De la pròpia definició de la constant de Lebesgue, s'entreveu que el la inestabilitat augmentarà a mesura que n creix. Concretament, considerant com fins ara nodes equiespaciats, quan $n \to \infty$, la constant de Lebesgue segueix la següent raó de creixement:

$$\Lambda_n \approx \frac{2^{n+1}}{e \cdot n \cdot \ln(x)}$$

i per tant, el problema es pot tornar inestable quan n es suficientment gran.

Un cop acabats tots els preliminars, ja es pot parlar de l'error. Primer introduirem un corol·lari del teorema (2.1.4), que ofereix una fita en funció del nombre de nodes del polinomi interpolador.

Corołari 2.1.11. *Sigui $E(x) = f(x) - P(x)$ l'error d'interpolació. Aquest compleix la cota següent:*

$$|E(x)| \leq \frac{\|f^{(n+1)}\|_\infty}{(n+1)!} \left| \prod_{j=0}^{n} (x - x_j) \right| \tag{2.1.5}$$

Demostració. Només s'ha de buscar aplicar la definició de norma del màxim, és a dir, fer el màxim:

$$|E(x)| = \left| \frac{f^{(n+1)}(\xi(x))}{(n+1)!} \right| \left| \prod_{j=0}^{n} (x - x_j) \right| \leq \max_{x \in I} \frac{\left| f^{(n+1)}(\xi(x)) \right|}{(n+1)!} \left| \prod_{j=0}^{n} (x - x_j) \right| =$$

$$= \frac{\|f^{(n+1)}\|_\infty}{(n+1)!} \left| \prod_{j=0}^{n} (x - x_j) \right|$$

□

I per acabar, el següent teorema descriu com de bé s'acosta un polinomi interpolador en uns nodes equispaciats donats a la funció que estem intentant interpolar.

Teorema 2.1.12. ***Estabilitat del Polinomi Interpolador de Lagrange:*** *Sigui $f \in \mathcal{C}^0([a,b])$ i $P(x)$ el seu polinomi interpolador relatiu als nodes $x_0, \ldots, x_n$. Llavors*

$$\|f - P\|_\infty \leq (1 + \Lambda_n) \inf_{Q \in \mathbb{P}_n} \|f - Q\|_\infty$$

Observació 2.1.13. L'expressió $\inf_{Q \in \mathbb{P}_n} \|f - Q\|_\infty$ representa el millor polinomi que aproxima la funció f, deixant doncs la fita màxima en funció de la millor aproximació possible i el nombre de condició. En altres paraules, l'error depen del nombre de nodes escollit i com de bé es pot aproximar la funció amb un polinomi.

Demostració. Sigui la següent funció que, donats el nodes d'interpolació $x_0, \ldots, x_n$, fa correspondre una funció el seu polinomi interpolador.

$$\begin{aligned} \mathcal{P} : \mathcal{C}([a,b]) &\longrightarrow \mathbb{P}_n \\ f &\longmapsto P \end{aligned}$$

Aquesta funció és lineal (vegis lema (2.1.14) demostrat després d'aquesta demostració) i compleix que si $f \in \mathbb{P}_n \Rightarrow \mathcal{P}(f) = f$ (el polinomi interpolador d'un polinomi és ell mateix).

Llavors, sigui $Q \in \mathbb{P}_n$ qualsevol. Sumant i restant Q a $f - P$ i aplicant la desigualtat triangular tenim:

$$|f(x) - P(x)| \leq |f(x) - Q(x)| + |Q(x) - P(x)| \tag{2.1.6}$$

A continuació desenvoluparem l'últim terme de la desigualtat $|Q(x) - P(x)|$ i usant la funció definida a l'inici d'aquesta demostració:

$$|Q(x) - P(x)| = |\mathcal{P}(Q(x)) - \mathcal{P}(f(x))| = |\mathcal{P}(Q - f)(x)|$$

usant a la última igualtat la linealitat de la funció. Seguint el desenvolupament, expressem la funció $Q - f$ com la imatge per $\mathcal{P}$, és a dir, com a polinomi interpolació de Lagrange:

$$|\mathcal{P}(Q-f)(x)| = \left|\sum_{i=0}^{n}(Q(x_i) - f(x_i))L_i(x)\right| \leq \sum_{i=0}^{n}|(Q(x_i) - f(x_i)| \cdot |L_i(x)| \leq$$

$$\leq \max_{x\in[a,b]}|Q(x) - f(x)|\sum_{i=0}^{n}|L_i(x)| \leq \|Q - f\|_\infty \sum_{i=0}^{n}|L_i(x)| \tag{2.1.7}$$

S'ha aconseguit expressar el terme $|Q(x) - P(x)|$ en funció de $\|Q - f\|$. Es vol ara expressar la desigualtat triangular (2.1.6) en termes de la norma del màxim, i es fa seguint la desigualtat amb el màxim

$$|f(x)-P(x)| \leq |f(x)-Q(x)|+|Q(x)-P(x)| \leq \max_{x\in[a,b]}|f(x)-Q(x)|+\max_{x\in[a,b]}|Q(x)-P(x)| \tag{2.1.8}$$

I com que s'ha desenvolupat l'últim terme en funció de la norma del màxim, substituint l'expressió a (2.1.7) obtenint:

$$\max_{x\in[a,b]}|Q(x) - P(x)| \leq \|Q - f\|_\infty \max_{x\in[a,b]}\sum_{i=0}^{n}|L_i(x)| = \|Q - f\|_\infty \Lambda_n$$

On recapitulant es veu:

$$|f(x)-P(x)| \leq \|f-Q\|_\infty+\max_{x\in[a,b]}|Q(x)-P(x)| \leq \|f-Q\|_\infty+\|Q-f\|_\infty\Lambda_n = (1+\Lambda_n)\|f-Q\|_\infty$$

Com que el polinomi $Q \in \mathbb{P}_n$ és arbitrari, també valdrà per l'ínfim, obtenint per tant el resultat que es volia provar.

$$|f(x) - P(x)| \leq (1 + \Lambda_n)\inf_{Q\in\mathbb{P}_n}\|f - Q\|_\infty$$

□

Lema 2.1.14. *La funció $\mathcal{P} : \mathcal{C}([a, b]) \longrightarrow \mathbb{P}_n$ que donats $n + 1$ nodes d'interpolació duu f al seu polinomi interpolador P a dits nodes, és lineal.*

Demostració. Per veure que és lineal s'ha de veure que:

1. $\mathcal{P}(f+g) = \mathcal{P}(f) + \mathcal{P}(g)$
2. $\mathcal{P}(\lambda f) = \lambda \mathcal{P}(f)$ tal que $\lambda \in \mathbb{R}$

$$1\text{: } \mathcal{P}(f+g) = \sum_{i=0}^{n} (f(x_i) + g(x_i)) L_i(x) = \sum_{i=0}^{n} f(x_i) L_i(x) + \sum_{i=0}^{n} g(x_i) L_i(x) = \mathcal{P}(f) + \mathcal{P}(g)$$

$$2\text{: } \mathcal{P}(\lambda f) = \sum_{i=0}^{n} \lambda f(x_i) L_i(x) = \lambda \sum_{i=0}^{n} f(x_i) L_i(x) = \lambda \mathcal{P}(f)$$

□

2.1.4 Limitacions del Mètode

Com s'ha ensenyat a l'apartat anterior, tot i la forma de Newton portar avantatges en l'avaluació del polinomi i en augmentar-ne el grau de manera computacionalment eficient, en cap moment s'ha mencionat que sigui un bon mètode d'aproximació si es manté la hipòtesi dels nodes equiespaciats. Quan el nombre de nodes (i el grau) augmenten, la aproximació al centre de l'interval millora moltíssim, però el desestabilitza i augmenta l'error de manera exponencial (recordem l'expressió de la constant de Lebesgue Λ_n) als extrems dels intervals, fent que no es cenyeixi gens a la funció que intenta interpolar, això és pot veure clarament amb l'anomenat contraexemple de Runge.

Com a petita desviació, cal mencionar que per arreglar el problema d'interpolació, només cal desfer-se de la hipòtesi de nodes equispaciats; el *polinomi de Txebixev*, on els nodes d'interpolació s'obtenen equispaciant-los en un cercle amb el diàmetre de la longitud del radi i projectant-los a l'interval d'aproximació, tal com es pot veure amb l'equació que donen els nodes:

$$x_k = \frac{a+b}{2} + \frac{b-a}{2}\hat{x}_k \quad \text{on} \quad \hat{x}_k = \cos\left(\frac{(2k+1)\pi}{2n+2}\right), k \in \mathbb{N}$$

Aquest mètode també elimina el problema del ràpid creixement del nombre de condició del polinomi de Lagrange, ja que la constant de Lebesgue obtinguda amb el polinomi de Txebixev segueix la següent raó de creixement:

$$\Lambda_n \approx \frac{\pi}{n+1} \ln(n)$$

que és una millora molt substancial, ja que s'ha passat d'un creixement exponencial a un logarítmic.

2.2 Mètode de la Potència

El mètode de la potència és un mètode iteratiu que permet trobar el valor propi de mòdul més gran i el seu vector propi associat.

Sigui $A \in \mathbb{K}^{n\times n}, \mathbb{K} = \mathbb{R}$ o $\mathbb{K} = \mathbb{C}$ una matriu diagonalitzable, i per tant, $\mathbb{K}^n$ té una base de vectors propis de A $x_1, \ldots, x_n$ Això permet en tot moment expressar $z \in \mathbb{K}^n$ com una combinació lineal d'aquests vectors propis:

$$z = \sum_{j=1}^{n} \alpha_j x_j \tag{2.2.1}$$

Definició 2.2.1. ***Valor Propi Dominant:*** *Siguin $\lambda_1, \ldots, \lambda_n$ valors propis de A. Diem que λ_1 és el valor propi dominant si compleix:*

$$|\lambda_1| > |\lambda_i|, i = 2, \ldots, n$$

És a dir, el valor propi dominant és el valor propi amb mòdul màxim.

Per parlar del mètode de la potència cal assumir que A té un valor propi dominant i, sense pèrdua de generalitat i per la comoditat del lector, també s'assumeix que els valors propis estan ordenats de major a menor modul però que no n'hi ha cap major o igual que el de mòdul màxim. És a dir, els valors propis compleixen:

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \ldots \geq |\lambda_n|$$

2.2.1 Idea i Intuïció

Consideris la matriu A anterior amb un únic valor propi dominant λ_1 i vectors propis unitaris linealment independents $x_1, \ldots, x_n$ amb valor propis associats $\lambda_1, \ldots, \lambda_n$. Consideris també la successió obtinguda de multiplicar un vector $z^{(0)} \in \mathbb{K}^n$ no nul recurrentment a una matriu:

$$z^{(k)} = \{z^{(0)}, z^{(1)} = Az^{(0)}, z^{(2)} = Az^{(1)}, \ldots\}$$

Primerament, s'ha de veure que la successió anterior pot ser representada expressant el terme k-èssim en funció de l'anterior. Això implica multiplicar la matriu k vegades per iteració per a calcular el terme k-èssim a partir del terme inicial:

$$z^{(k)} = Az^{(k-1)} = A(Az^{(k-2)}) = A^2 z^{(k-2)} = A^3 z^{(k-3)} = \ldots = A^k z^{(0)}$$

Tot i que computacionalment no és una transformació coherent ja que afegeix complexitat al calcular el terme k-èssim, és la propietat que ens permet deduir teòricament tots els resultat de convergència i la velocitat d'aquesta.

Usant (2.2.1) i que $Ax_i = \lambda_i x_i$, podem escriure:

$$z^{(k)} = A^k z^{(0)} = \sum_{j=1}^{n} \lambda_j^k \alpha_j x_j = \lambda_1^k \left[\alpha_1 x_1 + \sum_{j=2}^{n} \left(\frac{\lambda_j}{\lambda_1} \right)^k \alpha_j x_j \right] \tag{2.2.2}$$

Com que per hipòtesi A té un valor propi dominant, $\frac{|\lambda_j|}{|\lambda_1|} < 1$. Aleshores, dividint (2.2.2) entre λ_1^k a banda i banda:

$$\lim_{k \to \infty} \frac{z^{(k)}}{\lambda_1^k} = \alpha_1 x_1$$

És a dir, que la expressió anterior permet veure que la direcció del vector $z^{(k)}$ convergeix cap a la direcció de $\alpha_1 x_1$, cap al vector propi. Això ja dona una primera condició que apareixerà quan es formalitzi el Mètode de la Potència a la secció 2.2.2, que és que $\alpha_1 \neq 0$, sinó aquesta igualtat mai es podria donar. Si ens mirem més profundament la igualtat, en transpira que, tot i que la direcció de $z^{(0)}$ convergeix a la direcció del vector propi x_1, el modul del vector no té perquè fer-ho. S'il·lustra en el següent exemple exactament aquest fet.

Considerem $A \in \mathbb{R}^{2\times 2}$ amb vectors propis $v_1 = (1,0), v_2 = (0,1)$ i valors propis associats $\lambda_1 = 2, \lambda_2 = 1$ i $z^{(0)} = (1,1)$. Situant als eixos els vectors propis, s'ensenya a la Figura següent (2.1) els valors dels tres primers termes de la successió:

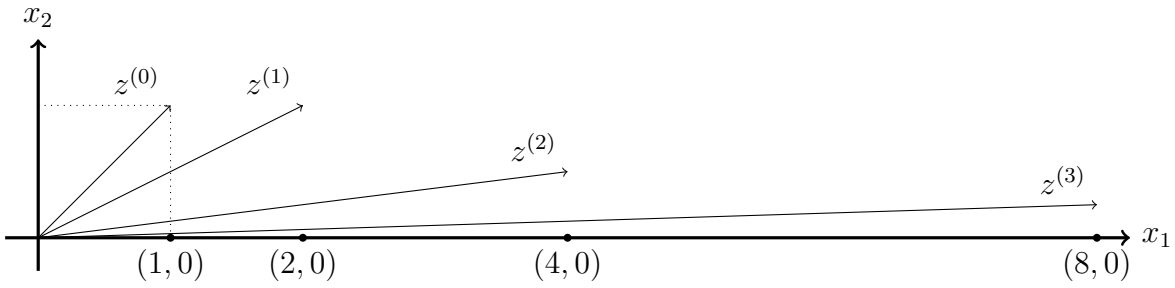


Figura 2.1: Convergència de la direcció de $z^{(k)}$ a x_1 però no del mòdul

La Figura 2.1 col·loca els eixos els vectors propis x_1, x_2. Clarament es veu com $z^{(i)}$ es va acostant a l'eix a mesura que i augmenta, però el seu mòdul creix amb raó 2^i. És a dir, la direcció de $z^{(k)}$ quan creix va cap a la direcció del vector propi, però el mòdul convergeix a infinit.

En aquest exemple el mòdul tendeix a infinit, però si s'escullen els valors propis complint que $|\lambda_2| < |\lambda_1| < 1$, succeiria que el modul convergiria cap a 0 a mesura que k augmentés. Cap de les dues opcions es comporta bé numèricament, ja que pot donar problemes d'*underflow* o *overflow* i tampoc s'està aconseguint el valor del vector propi, sinó un múltiple d'aquest.

La solució intuïtiva a aquest problema és normalitzar el vector a cada iteració per evitar que el seu mòdul creixi, i es veurà després al lema 2.2.4 que efectivament la solució intuïtiva és la que aconsegueix la convergència. Abans de passar a la formalització, es mostra a la Figura 2.2 què ocorre amb el vector si és normalitzat a cada iteració.

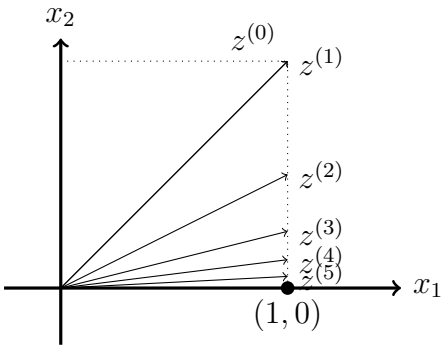


Figura 2.2: Convergència de $z^{(k)}$ a x_1

És a dir, $z^{(k)} \overset{k\to\infty}{\longrightarrow} x_1$ tan en direcció com en modul.

2.2.2 Formalització

Un cop s'ha vist intuïtivament que el mètode descrit convergirà, procediram a formalitzar el mètode de la potència i demostrar que el vector d'inici $z^{(k)}$ propi convergirà amb raó $(|\lambda_2|/|\lambda_1|)^k$ a x_1. Això es resultat de l'elecció d'escollir els valors propis ordenats, sinó seria amb raó $(|\lambda_j|/|\lambda_1|)^k$, on j és l'índex que maximitza el mòdul del valor propi exceptuant el primer.

Definició 2.2.2. ***Mètode de la Potència:*** *Sigui $A \in \mathbb{K}^{n\times n}, \mathbb{K} = \mathbb{R}$ o $\mathbb{K} = \mathbb{C}$ una matriu diagonalitzable. Escollint un $q^{(0)} \in \mathbb{K}^n$ unitari ($\|q^{(0)}\|_2 = 1$), denominem a les següents iteracions el Mètode de la potència:*

$$\begin{aligned} z^{(k)} &= Aq^{(k-1)} \\ q^{(k)} &= \frac{z^{(k)}}{\|z^{(k)}\|_2} \\ \nu^{(k)} &= (q^{(k)})^H A q^{(k)} \end{aligned} \tag{2.2.3}$$

On $q^{(0)} = \sum_{j=1}^{n} \alpha_j x_j$ amb $\alpha_1 \neq 0$, on $x_1, \ldots, x_n$ és una base de vectors propis associada als valors propis $\lambda_1, \ldots, \lambda_n$.

Observació 2.2.3. Recodis que z^H és l'Hermitiana de z. Si $z \in \mathbb{C}^{n\times n}$, z^H es refereix a la transposada conjugada de $z (z^H = \bar{z}^T)$. Així, per a $x \in \mathbb{R}^{n\times n}$, $x^H = x^T$.

El primer que s'ha de veure és que la normalització del vector z a cada iteració normalitza efectivament la successió. Es el resultat del següent lema:

Lema 2.2.4. *Donada la matriu A amb les característiques anteriors, es compleix:*

$$q^{(k)} = \frac{A^k q^{(0)}}{\|A^k q^{(0)}\|}, k \geq 0$$

És a dir, la successió $q^{(k)}$ queda normalitzada.

Demostració. Procedim per inducció, on el cas base és trivial. El cas inductiu es desenvolupa de la següent manera:

$$q^{(k)} = \frac{Aq^{(k-1)}}{\|Aq^{(k-1)}\|} = \frac{A^k q^{(0)}}{\|A^{k-1} q^{(0)}\|} \frac{\|A^{k-1} q^{(0)}\|}{\|A^k q^{(0)}\|} = \frac{A^k q^{(0)}}{\|A^k q^{(0)}\|}$$

□

Aquesta expressió permet expressar $(q^{(k)})$ com a (2.2.2) usant que A és diagonalitzable, on la successió efectivament segueix convergint.

Un cop provat que expressar el vector k-èssim com $A^k q^{(0)}$ normalitza, veiem que efectivament el vector inicial $q^{(0)}$ convergeix cap al vector propi en la següent proposició.

Teorema 2.2.5. ***Convergència al Valor Propi:*** *Sigui $A \in \mathbb{K}^{n \times n}$ una matriu diagonalitzable amb un valor propi dominant λ_1. Sigui $q^{(0)} = \sum_{j=1}^{n} \alpha_j x_j$ amb $\alpha_0 \neq 0$ on $x_1, \ldots, x_n$ és una base de vectors propis de modul unitari amb $\lambda_1, \ldots, \lambda_n$ valors propis associats. Llavors:*

$$\|\tilde{q}^{(k)} - x_1\|_2 \leq C \left| \frac{\lambda_2}{\lambda_1} \right|^k, k \geq 1$$

on

$$\tilde{q}^{(k)} = \frac{q^{(k)} \|A^k q^{(0)}\|_2}{\alpha_1 \lambda_1^k}$$

Demostració. El primer pas de la demostració és aclarir qui és $\tilde{q}$ i per a què es vol emprar en comptes del vector $q^{(k)}$ directament. Primer expliquem la idea darrera l'expressió.

Idea: Volem expressar el vector $q^{(k)}$ com a (2.2.2) però amb el coeficient del vector propi associat al valor propi x_1 igual a 1, ja que per veure la convergència s'usarà una base de valors propis associada (sense pèrdua de generalitat) que tingui mòdul 1 ($\|x_i\|_2 = 1, i = 1, \ldots, n$) per veure que la convergència és efectivament al vector propi. Per aconseguir-ho, es divideix l'expressió (2.2.2) per $\alpha_1 \lambda_1^k$:

$$\left[x_1 + \sum_{j=2}^{n} \frac{\alpha_j}{\alpha_1} \left(\frac{\lambda_j}{\lambda_1} \right)^k x_j \right] = \frac{A^k q^{(0)}}{\alpha_1 \lambda_1^k} = \frac{q^{(k)}}{\alpha_1 \lambda_1^k} \tag{2.2.4}$$

I concretament, com que normalitzem el vector entre iteració i iteració, i per tant $\|A^k q^{(0)}\|_2$ apareix pel lema 2.2.4. Operant arribem a la definició de $\tilde{q}^{(k)}$:

$$q^{(k)} = \frac{A^k q^{(0)}}{\|A^k q^{(0)}\|} \Leftrightarrow q^{(k)} \|A^k q^{(0)}\| = A^k q^{(0)} \Leftrightarrow \frac{q^{(k)} \|A^k q^{(0)}\|}{\alpha_1 \lambda_1^k} = \frac{A^k q^{(0)}}{\alpha_1 \lambda_1^k} =$$

$$= x_1 + \sum_{j=2}^{n} \frac{\alpha_j}{\alpha_1} \left(\frac{\lambda_j}{\lambda_1} \right)^k x_j = \tilde{q}^{(k)}$$

que és el que hem obtingut a (2.2.4). Ara, acabem la demostració.

Al ser A diagonalitzable, es pot escollir una matriu no singular X que compleixi $\|x_i\|_2 = 1, i = 1, \ldots, n$ associada a A sense pèrdua de generalitat. Aleshores:

$$\|x_1 + \sum_{j=2}^{n} \frac{\alpha_j}{\alpha_1} \left(\frac{\lambda_j}{\lambda_1} \right)^k x_j - x_1\|_2 = \left\| \sum_{j=2}^{n} \frac{\alpha_j}{\alpha_1} \left(\frac{\lambda_j}{\lambda_1} \right)^k x_j \right\|_2 \leq \left(\sum_{j=2}^{n} \left[\frac{\alpha_j}{\alpha_1} \right]^2 \left[\frac{\lambda_j}{\lambda_1} \right]^{2k} \right)^{1/2} \leq$$

$$\leq \left| \frac{\lambda_2}{\lambda_1} \right|^k \left(\sum_{j=2}^{n} \left(\frac{\alpha_j}{\alpha_1} \right)^2 \right)^{1/2} = \left| \frac{\lambda_2}{\lambda_1} \right|^k C$$

que és l'expressió del teorema que es volia provar amb el valor de $C = (\sum_{i=0}^{n} (\alpha_j/\alpha_1)^2)^{1/2}$
□

Una dels grans avantatges del mètode de la potència respecte altres mètodes de calcul de vectors propis o valors propis és que podem calcular el vector propi i el valor propi associat en la mateixa iteració, que és la tercera operació de la iteració. Aquesta operació és l'anomenat quocient de Rayleigh

Definició 2.2.6. ***Quocient de Rayleigh:*** *Sigui $z \in \mathbb{K}^n$ i A una matriu diagonalitzable. Definim el Quocient de Rayleigh com:*

$$\nu_k = \frac{z^H A z}{z^H z}$$

Observació 2.2.7. Si x és un vector propi de A i recordant que $Ax = \lambda x$, tenim:

$$\nu = \frac{x^H A x}{x^H x} = \frac{\lambda x^H x}{x^H x} = \lambda$$

Per tant, el quocient de Rayleigh serveix per obtenir el valor propi per la seva propia definició.

A la següent proposició es prova que la successió ν convergeix cap a λ_1 amb el mateix ordre que el vector propi.

Proposició 2.2.8. ***Convergència al Valor Propi Dominant:*** *Amb les mateixes hipòtesi del teorema anterior, la iteració de Rayleigh convergeix al valor propi dominant de la següent manera:*

$$\nu^{(k)} = \lambda_1 + O\left(\frac{|\lambda_2|}{|\lambda_1|}\right)^k$$

Demostració. Usant l'expressió de $\tilde{q}^{(k)}$ de l'expressió anterior i escrivint el quocient de Rayleigh amb ella es veu:

$$\frac{(\tilde{q}^{(k)})^H A\tilde{q}^{(k)}}{\|\tilde{q}^{(k)}\|_2^2} = \frac{\left(\dfrac{q^{(k)}\|A^k q^{(0)}\|_2}{\alpha_1\lambda_1^k}\right)^H A\,\dfrac{q^{(k)}\|A^k q^{(0)}\|_2}{\alpha_1\lambda_1^k}}{\left\|\dfrac{q^{(k)}\|A^k q^{(0)}\|_2}{\alpha_1\lambda_1^k}\right\|_2^2} = \frac{\left(q^{(k)}\right)^H Aq^{(k)}}{\|q^{(k)}\|_2^2} = \left(q^{(k)}\right)^H Aq^{(k)} = \nu^{(k)}$$

On es cancel·len tots els coeficients, i la norma de $q^{(k)}$ és 1 ja que aquesta està normalitzada per definició del mètode de la potència. Tal com hem vist a l'observació, el quocient de Rayleigh convergeix cap al valor propi associat al vector propi que usem pel producte, per tant:

$$\lim_{k\to\infty} \nu^{(k)} = \lambda_1$$

amb exactament el mateix ordre que ho fa el vector propi associat, que és $\left|\dfrac{\lambda_2}{\lambda_1}\right|^k$. □

En relació a aquesta última demostració, si la matriu A no només és diagonalitzable sinó que també és simètrica, es pot escollir una base de vectors propis $x_1, \ldots, x_n$ ortonormals, fet que dobla la velocitat de convergència del mètode:

Proposició 2.2.9. *Sigui $A \in \mathbb{K}^{n\times n}$ diagonalitzable i simètrica i les hipòtesi del Mètode de la Potència. Aleshores, el quocient de Rayleight convergeix cap el valor propi dominant amb raó $\left|\frac{\lambda_2}{\lambda_1}\right|^{2k}$. Concretament:*

$$\nu^{(k)} = \lambda_1 + O\left(\frac{\lambda_2}{\lambda_1}\right)^{2k}$$

Demostració. Comencem expressant el quocient de Rayleigh en funció de z, sense normalitzar i expressant la norma com a producte escalar.

$$\nu^{(k)} = \frac{(z^{(k)})^H Az^{(k)}}{(z^{(k)})^H z^{(k)}} = \frac{\left(\sum_{j=1}^n \lambda_j^k \alpha_j x_j^H\right)\left(\sum_{j=1}^n \lambda_j^{k+1}\alpha_j x_j\right)}{\left(\sum_{j=1}^n \lambda_j^k \alpha_j x_j^H\right)\left(\sum_{j=1}^n \lambda_j^k \alpha_j x_j\right)}$$

Al ser la matriu simètrica i, per tant, la base de vectors propis ortonormal, el producte de vectors de la base es comporta com una delta de Kronecker:

$$x_i^H x_j = \begin{cases} 1 & \text{si } i = j \\ 0 & \text{si } i \neq j \end{cases}$$

Fent doncs el producte dels dos sumatoris i expressant-los com (2.2.2) obtenim la següent expressió:

$$\nu^{(k)} = \frac{\sum_{j=1}^n \alpha_j^2 \lambda_j^{2k+1}}{\sum_{j=1}^n \alpha_j^2 \lambda_j^{2k}} = \frac{\lambda_1^{2k+1}\left[\alpha_1^2 + \sum_{j=2}^n \alpha_j^2 \left(\frac{\lambda_j}{\lambda_1}\right)^{2k+1}\right]}{\lambda_1^{2k}\left[\alpha_1^2 + \sum_{j=2}^n \alpha_j^2 \left(\frac{\lambda_j}{\lambda_1}\right)^{2k}\right]} = \lambda_1 \frac{1 + \sum_{j=2}^n \left(\frac{\alpha_j}{\alpha_1}\right)^2 \left(\frac{\lambda_j}{\lambda_1}\right)^{2k+1}}{1 + \sum_{j=2}^n \left(\frac{\alpha_j}{\alpha_1}\right)^2 \left(\frac{\lambda_j}{\lambda_1}\right)^{2k}}$$

On en l'última igualtat s'ha dividit el numerador i el denominador entre $\lambda^{2k}\alpha_1^2$. Per tant, s'ha de veure doncs el valor que pren la fracció quan $k \to \infty$. Obviant la λ_1 de moment, separant la fracció es s'observa clarament que aplicant la raó geomètrica $\frac{1}{1+x} = 1 - x + x^2 + \ldots$ s'obté la següent expressió:

$$\left[1 + \sum_{j=2}^n \left(\frac{\alpha_j}{\alpha_1}\right)^2 \left(\frac{\lambda_j}{\lambda_1}\right)^{2k+1}\right] \left[\frac{1}{1 + \sum_{j=2}^n \left(\frac{\alpha_j}{\alpha_1}\right)^2 \left(\frac{\lambda_j}{\lambda_1}\right)^{2k}}\right] = \left[1 + \sum_{j=2}^n \left(\frac{\alpha_j}{\alpha_1}\right)^2 \left(\frac{\lambda_j}{\lambda_1}\right)^{2k+1}\right] \cdot$$

$$\cdot \left[1 - \sum_{j=2}^n \left(\frac{\alpha_j}{\alpha_1}\right)^2 \left(\frac{\lambda_j}{\lambda_1}\right)^{2k} + \left(1 + \sum_{j=2}^n \left(\frac{\alpha_j}{\alpha_1}\right)^2 \left(\frac{\lambda_j}{\lambda_1}\right)^{2k}\right)^2 + \ldots\right] =$$

$$= \left[1 - \sum_{j=2}^n \left(\frac{\alpha_j}{\alpha_1}\right)^2 \left(\frac{\lambda_j}{\lambda_1}\right)^{2k} + \left(1 + \sum_{j=2}^n \left(\frac{\alpha_j}{\alpha_1}\right)^2 \left(\frac{\lambda_j}{\lambda_1}\right)^{2k}\right)^2 + \ldots\right]$$

On en l'última igualtat s'ha emprat que el terme dominant, que per $|\lambda_2|/|\lambda_1| < 1$, és el que té el grau més petit, que és la raó geomètrica. Aplicant el mateix argument, el terme dominant quan $k \to \infty$ serà el que té el grau menor, i recordant que s'ha de multiplicar per λ_1 obtenim:

$$\nu^{(k)} = \lambda_1 \left[1 - \sum_{j=2}^n \left(\frac{\alpha_j}{\alpha_1}\right)^2 \left(\frac{\lambda_j}{\lambda_1}\right)^{2k}\right] = \lambda_1 + O\left(\frac{|\lambda_2|}{|\lambda_1|}\right)^{2k}$$

i per tant hem vist el que demanava l'enunciat. □

Observació 2.2.10. La demostració de la proposició 2.2.8 també es podria fer seguint l'aproximació vista a la demostració anterior, tot i que obtenint uns càlculs molt més farragosos ja que sense la hipòtesi d'ortonormalitat a la base, s'ha de destriar el terme dominant d'entre molts altres termes. Per això s'ha decidit no fer-la així, però si fer la de la proposició anterior per mostrar-ho.

Tant en les proposicions (2.2.8) i (2.2.9) com en el teorema (2.2.5) s'ha emprat la norma Euclidiana. És natural preguntar-se què succeeix si intentem descriure el Mètode de la Potència però amb una altra norma, i concretament amb la norma $\|\cdot\|_\infty$ es troba un comportament digne de menció.

La norma del màxim comporta al mètode de la potència una avantatge respecte a fer-lo amb la norma euclidiana:

Definició 2.2.11. ***Mètode de la Potència*** $\|\cdot\|_\infty$***:*** *Sota les mateixes hipòtesi de la definició 2.2.2 però considerant la norma del màxim, anomenem a la següent iteració Mètode de la Potència:*

$$\begin{aligned} z^{(k)} &= Aq^{(k-1)} \\ q^{(k)} &= \frac{z^{(k)}}{\|z^{(k)}\|_\infty} \\ \mu^{(k)} &= z_p^{(k)} = \sigma_k \|z^{(k)}\|_\infty \end{aligned} \tag{2.2.5}$$

on p és la component del vector $z^{(k)}$ tal que el valor és el màxim i σ_k és el signe de $z_p^{(k)}$.

Com es veu, no és necessari calcular el quocient de Rayleigh a cada iteració, ja que (com es demostrarà més endavant a la proposició (2.2.12)) amb només el valor més gran del vector $z^{(k)}$, la successió $\mu^{(k)} \overset{k\to\infty}{\longrightarrow} \lambda_1$. Per tant, l'ús de la norma del màxim redueix el nombre d'operacions per iteració, ja que és molt menys costós trobar el màxim que no calcular el quocient de Rayleigh. A canvi, s'ha de saber el signe del component més gran del vector, si no es té en compte el mètode només convergiria a valors propis dominants positius.

La demostració de la convergència del vector propi, tot i haver estat enunciada i provada amb la norma euclidiana, no depen de la norma escollida, només varia la constant C. Tampoc fa falta reescriure-la tenint en compte el signe que s'ha d'extreure a la successió $\mu^{(k)}$, ja que amb la divisió de $\alpha_1\lambda_1$, sempre queda el signe positiu independement de la norma.

En canvi, s'ha trobat interessant fer la demostració conforme la successió de les components de valor més alt.

Proposició 2.2.12. ***Convergència al Valor Propi Dominant:*** *Considerem A sota les mateixes hipòtesis que la definició del mètode de la potència, anàlogament amb $z^{(0)}$ però amb $q^{(0)} = \frac{z^{(0)}}{\|z^{(0)}\|_\infty}$. Aleshores, la successió $\mu^{(k)}$ convergeix cap al valor propi dominant amb raó $\left|\frac{\lambda_2}{\lambda_1}\right|^k$. Concretament:*

$$\mu^{(k)} = \lambda_1 + O\left(\frac{\lambda_2}{\lambda_1}\right)^k$$

Demostració. Per provar aquest resultat, es desenvolupa la definició així:

$$\mu^{(k)} = \sigma_p z_p^{(k)} = \sigma_p \frac{z_{p_k}^{(k)}}{1} = \sigma_p \frac{z_p^{(k)}}{q_p^{(k-1)}}$$

On s'ha usat que el vector $q^{(k)}$ està normalitzat, la seva component de mida més gran serà 1. Cal notar que l'índex p pot canviar per cada iteració, però per norma general es fixa quan k és prou gran. Per tant, per comoditat del lector i sent conscients de l'abús de notació, s'ha emprat p per referir-nos (potser) a components diferents.

Si es reescriu l'expressió com a (2.2.2) i dividint tots els termes entre α_1, es pot desenvolupar com:

$$\mu^{(k)} = \frac{\lambda_1^k \sigma_p \left[x_1 + \sum_{j=2}^n \frac{\alpha_j}{\alpha_1} \left(\frac{\lambda_j}{\lambda_1} \right)^k x_j \right]_p}{\lambda_1^{k-1} \sigma_p \left[x_1 + \sum_{j=2}^n \frac{\alpha_j}{\alpha_1} \left(\frac{\lambda_j}{\lambda_1} \right)^{k-1} x_j \right]_p} = \lambda_1 \frac{\left[x_1 + \sum_{j=2}^n \frac{\alpha_j}{\alpha_1} \left(\frac{\lambda_j}{\lambda_1} \right)^k x_j \right]_p}{\left[x_1 + \sum_{j=2}^n \frac{\alpha_j}{\alpha_1} \left(\frac{\lambda_j}{\lambda_1} \right)^{k-1} x_j \right]_p}$$

I de la mateixa manera que a la proposició (2.2.8), hem de calcular quan $k \to \infty$, i pel mòdul del quocient dels valors propis, sabem que el terme de grau més baix és el que dominarà el creixement, i per tant:

$$\mu^{(k)} = \lambda_1 + O\left(\frac{|\lambda_2|}{|\lambda_1|} \right)^k$$

Que és el que es volia provar. □

2.2.3 Problemes d'Implementació i Limitacions

En aquest apartat es vol parlar de diverses limitacions del Mètode de la potència i de diverses solucions associades.

Hem vist al teorema (2.2.5) que la raó de convergència del teorema és $\left| \frac{\lambda_2}{\lambda_1} \right|^k$ i que tot el mètode assumeix que hi ha un valor propi dominant. Fent una altra ullada a la raó, veiem que el mètode serà especialment efectiu si el quocient entre λ_1 i λ_2 és suficientment gran. En canvi, succeeixen tres casos quan tenim dos valors propis dominants, és a dir, si $|\lambda_1| = |\lambda_2|$:

1. $\lambda_1 = \lambda_2$: Si els dos valors propis dominants coincideixen, el mètode segueix convergint. Això es pot veure recuperant l'expressió de (2.2.2) i considerant que hi ha dos

vectors d'igual mòdul, reescrivint-la:

$$z^{(k)} = \lambda_1^k \left[\alpha_1 x_1 + \alpha_2 x_2 + \sum_{j=3}^{n} \left(\frac{\lambda_j}{\lambda_1} \right)^k \alpha_j x_j \right]$$

Com que $\alpha_1 x_1 + \alpha_2 x_2$ segueix sent una combinació lineal de vectors propis de A el mètode convergirà, i per tant per la proposició 2.2.8, $\nu^{(k)}$ convergirà a λ_1.

2. $\lambda_2 = -\lambda_1$: si els dos valors propis dominants tenen signe contrari, el mètode convergeix aplicant-lo a A^2, ja que $\lambda_i(A^2) = [\lambda_i(A)]^2 \Rightarrow \lambda_1^2 = \lambda_2^2$, on tornem a tenir el mateix cas descrit a l'ítem 1.

3. $\lambda_2 = \bar{\lambda_1}$. Si els dos valors propis dominants són complexos conjugats, el mètode no convergeix, ja que $q^{(k)}$ itera sobre els vectors propis associats. La intuïció darrera d'aquest cas és que l'aproximació va rotant.

En relació als problemes d'implementació, ja hem mencionat que la normalització a cada iteració és necessària per evitar un *underflow* o *overflow* [3] aritmètic. L'altre fet relacionat amb l'implementació està relacionat amb la hipòtesi $\alpha_1 \neq 0$, hipòtesi que en implementacions pot no ser necessària. En molts casos, errors d'arrodoniment inevitables fan que a la pràctica α_1 sigui diferent de zero, i que per tant el mètode es pogui fer, tot i que el valor d'α_1 sigui molt petit.

[3] Referint-nos que els valor de la variable és massa gran com per ser emmagatzemat a memòria o massa proper a zero

Capítol 3

Conceptes Relacionats

Aquest capítol introdueix certs conceptes i tecnologies necessaris que poden facilitar la lectura i comprensió d'aquest escrit. Per començar, s'explicarà què és un Agent Conversacional i quins marcs es poden usar per dissenyar i implementar-los. Seguidament, s'introduirà la tecnologia Docker, que facilita la construcció i desplegament d'aplicacions lleugeres amb el menor nombre de dependències possible. Per acabar, es parlarà sobre *Webhooks* en el disseny d'aplicacions web.

3.1 Marcs Conversacionals

Els marcs conversacionals (o *chatbot frameworks*), són un conjunt d'eines i llibreries de programació que faciliten la creació de bot de conversa. Aquestes plataformes serveixen per dissenyar, construir i desplegar *xatbots* que poden interactuar amb els usuaris en llenguatge natural, i són utilitzats en una gran varietat d'indústries, com ara el servei d'atenció al client, el màrqueting o els assistents virtuals.

Existeixen una gran varietat de marcs conversacionals, els més populars sent DialogFlow [25] de Google, Watson Assistant[26] d'IBM i Rasa Open Source. Tots aquests marcs, tot i que tenen les seves pròpies característiques i funcions específiques, comparteixen certs trets fonamentals comuns.

El primer concepte transversal és la intenció (o *intent*) de l'usuari, que és l'objectiu que té l'usuari quan interactua amb l'agent. Per exemple, si un usuari diu 'Quan és l'examen de programació?' la intenció que ho defineixi pot ser 'quan és l'examen de programació'.

El segon concepte important són les entitats (o *entities*), que són les paraules clau dins d'una conversa que l'agent necessita per comprendre el context i proporcionar una resposta precisa. En l'exemple anterior, 'programació' podria ser l'entitat 'assignatura'.

El tercer és la memòria contextual, que permet l'agent mantenir el context de la conversació al llarg de múltiples intercanvis amb l'usuari. Això li permet recordar certs detalls o entitats i no haver-les de preguntar a l'usuari, construint així una experiència més natural per a l'usuari.

El quart i últim, el gestor de diàleg (o *dialogue management*) és el conjunt de mecanismes que controlen el flux de la conversa i decideix quina serà la pròxima resposta de l'agent basant-se en el context actual, les intencions i les entitats reconegudes.

Una altra característica comuna a tots aquests marcs són les tècniques d'aprenentatge automàtic. Aquestes permeten als agents entendre millor el llenguatge natural, identificar patrons subtils a les dades i en aprendre dels mateixos usuaris i les seves interaccions. Aquestes capacitats milloren la comprensió de l'agent, li permeten adaptar-se a diferents estils de llenguatge i millorar la seva precisió en la identificació d'intencions i entitats.

No obstant això, la implementació i gestió d'agents basats en aprenentatge automàtic no estan absents de reptes. Es necessita una gran quantitat de dades etiquetades per a l'entrenament, cosa que pot implicar una gran inversió de temps i recursos. A més, l'ajustament i la interpretació dels models d'aprenentatge automàtic poden ser complexos, requerint habilitats i coneixements especialitzats.

Tot i la clara superioritat d'aquests marcs conversacionals basats en aprenentatge automàtic, els bots de conversa que es troben en gran part del web segueixen usant tecnologies basades en patrons, com els d'AIML *(Artificial Intelligence Markup Language)*, ja que no requereixen dades d'entrenament i són més senzills d'implementar.

3.2 Docker

Docker[3] és una plataforma oberta per desenvolupar, empaquetar i executar aplicacions. Concretament, Docker permet empaquetar una aplicació i les seves dependències en un entorn anomenat contenidor, que està aïllat del sistema operatiu. Els contenidors són lleugers i contenen només el que l'aplicació necessita per executar-se, evitant haver d'instal·lar les llibreries o dependre de quines llibreries o programari té la màquina local. Addicionalment, una màquina local pot córrer múltiples contenidors sense tenir problemes de rendiment, ja que són molt lleugers i s'executen mitjançant els recursos del sistema operatiu no creant els seus.

En altres paraules, un contenidor de Docker és una capsa aïllada del sistema operatiu que conté totes les dependències necessàries a nivell de llibreries, però no conté cap estructura ni funcionalitat per executar el programari dins seu com si ho tindria un sistema operatiu. Una bona analogia seria veure un Docker com una màquina virtual - està dins del sistema operatiu i està aïllada d'aquest - però sense la redundància dels recursos que aquesta obliga a generar. És a dir, una màquina virtual de Linux executant-se en un Linux té dos

nuclis, el nucli de la màquina virtual i el nucli de la màquina local. En canvi, el contenidor no ha de crear cap recurs que pogui proporcionar sistema operatiu, ja que mitjançant un *socket*[1] li fa peticions perquè executi les instruccions que emmagatzema, per això és molt més lleuger i eficient.

La lleugeresa i la flexibilitat de les imatges s'aconsegueix amb l'estructura mostrada a la Figura 3.1. Amb el que el programador interactua és amb el Client de Docker, el qual fa peticions al *Docker Daemon*, que és el que interactua amb les imatges i crea els contenidors. Els contenidors, doncs són instàncies concretes de les imatges. Un exemple seria veure la imatge com els plànols d'una casa, mentre que el contenidor serien les diverses cases de la promoció construïdes arreu seguint els plànols.

Una imatge és un conjunt d'instruccions que s'usen per crear un contenidor. Generalment, una imatge de les que s'ensenyen a la Figura 3.1, són imatges prefetes contenint una sèrie de dependències i programari pensat per una tasca específica. Per exemple, la imatge de Python conté totes les dependències necessàries per a l'execució de Python. N'hi ha d'oficials i de generades per la comunitat (que s'obtenen a través del registre de Docker tal com es mostra a la Figura 3.1, el rectangle de més a la dreta) que empren imatges ja fetes per afegir-hi noves funcionalitats o adaptar-les als requisits necessaris, com és el cas d'aquest projecte, on pels mòduls de l'aplicació s'ha utilitzat imatges prefetes de Java, C, Python modificant-les perquè executessin els programes dissenyats per l'aplicació o s'han hem emprat directament per MariaDB o per Rasa.

[1]Socket: concepte abstracte mitjançant el qual dos programes poden fluxos de dades de manera fiable i ordenada.

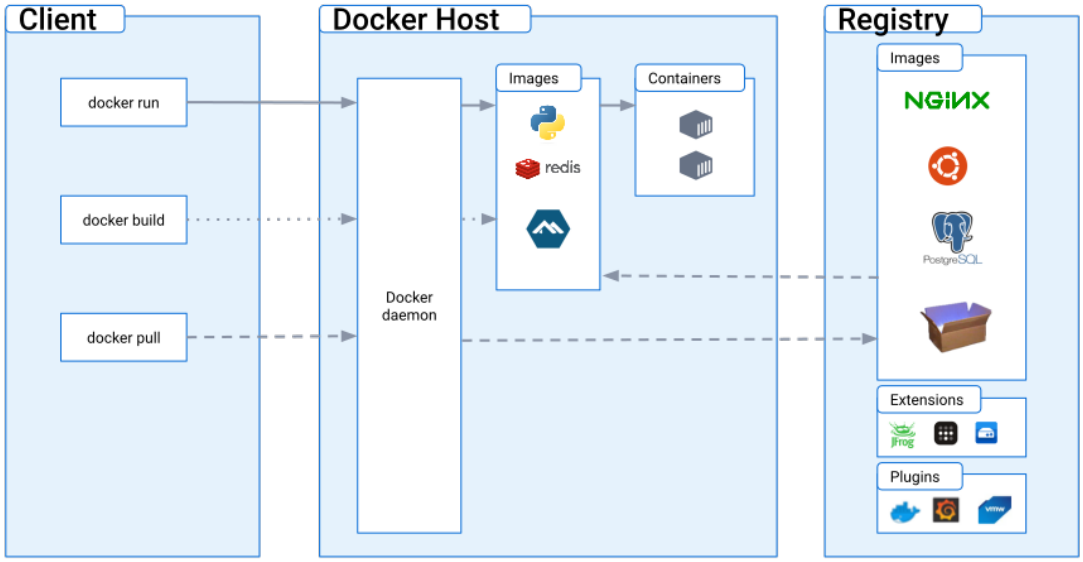


Figura 3.1: Estructura de la plataforma de Docker en l'execució.

Els avantatges de Docker són sobretot pel que fa a desplegament i aïllament de responsabilitats al codi. Els seus principis estan molt alineats amb els principis de responsabilitat única - és a dir, cada contenidor, una funcionalitat - fent arquitectures de projectes més netes i més ampliables. Amb l'ús de Docker-Compose (un client de Docker) es poden aixecar múltiples contenidors al mateix temps, i els contenidors en si mateixos són portables i depenent molt poc de la màquina local que els usa, permetent que els programadors es puguin centrar en el desenvolupament de l'aplicació i que en el moment del seu desplegament, sense que els programadors s'hagin de preocupar en quina màquina s'executarà el seu codi. Per aquesta raó s'han *dockeritzat* tots els mòduls que conformen aquest projecte.

3.2.1 Volums

Els volums[4] són un mecanisme de persistència de dades generat i usat pels contenidors. La necessitat dels volums sorgeix per l'absència de persistència de dades dels contenidors, que en acabar l'execució i ser aturats eliminen totes les dades emprades i generades durant aquestes, privant al programador del seu ús posterior. També, pel funcionament del mecanisme dedicat a solucionar aquest problema, es poden fer servir per compartir fitxers entre la màquina local i el contenidor. En concret, en aquest projecte s'ha usat un volum per obtenir els *logs* de les converses de l'usuari amb el bot (vegeu secció 8.4.3) i un altre per subministrar el contenidor amb la implementació de l'alumne dels enunciats i els tests, de manera que el contenidor els pugui executar (vegeu secció 7.6.2).

Un Volum és un sistema d'emmagatzematge separat del contenidor i emmagatzemat a la màquina local, concretament a una part on Docker té permisos. Quan es crea un contenidor, s'hi pot adjuntar (*attach*) el volum, i quan el procés del contenidor escriu dades a una carpeta preassignada de dins contenidor, Docker envia una petició al sistema operatiu perquè també escrigui les dades a la ubicació designada pel contenidor.

La Figura 3.2 mostra un diagrama de com funciona el volum. Quan s'ha adjuntat un volum a un contenidor, s'associa un (*bind*) a un directori concret de la màquina local, el *Filesystem* del dibuix, amb un directori del contenidor. Aleshores, quan el procés del contenidor escriu al directori del contenidor, l'escriptura també es realitza al quadrat taronja del dibuix, la *Docker Area*, que és dins de la màquina local i no del contenidor, proporcionant una manera d'extreure dades de dins del contenidor.

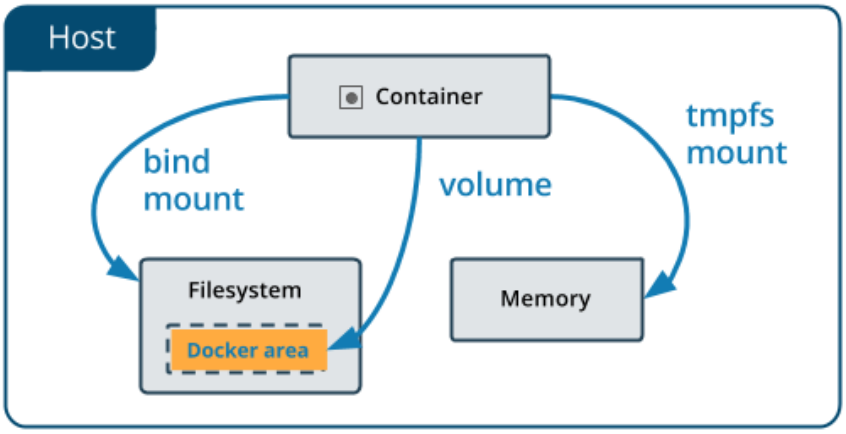


Figura 3.2: Estructura de compartir un volum de Docker

Aquest sistema també es pot usar a la inversa: en comptes de què el contenidor escrigui a la màquina local, que la màquina local proporcioni fitxers al contenidor. Com que es vinculen l'un a l'altre, si a l'associar (*bind*) s'escull un directori de la màquina local amb continguts, aquests són accessibles pel contenidor.

3.3 Webhooks

En el context del desenvolupament d'aplicacions web, les API[2] actuen com a pont entre dues aplicacions de programari, permetent que interactuïn entre si. Aquestes interaccions es fan mitjançant peticions: una aplicació sol·licita alguna informació o acció a través de l'API, i l'API respon amb el resultat desitjat. Aquest mètode és increïblement eficient, ja que l'API només ha de processar les peticions quan arriben. Però, què passa quan una

[2]Interfície de Programació d'Aplicacions o *Application Programming Interface*

aplicació necessita estar al corrent d'un esdeveniment a l'altra aplicació, en particular sense haver de fer peticions constants a l'API?

Els *Webhooks* són peticions asíncrones en temps real disparades per un esdeveniment que activen una funció concreta a la màquina que les rep. Posem un exemple d'aquest projecte: quan un usuari envia a Telegram un missatge, s'ha de generar una resposta. Telegram doncs dispara el *webhook*, contenint aquest la funció que l'API ha d'activar i les dades associades que podrien ser necessàries, en el nostre exemple, la funció cridaria al mòdul de Rasa i les dades serien l'identificador de l'usuari i el missatge que ha enviat.

Al contrari que les peticions a una API, als *Webhooks* les dades no flueixen en dues direccions, sinó només en una i fa actuar el servidor de maneres predefinides. L'alternativa a usar *webhooks* per aquest tipus de transmissió de dades és fer una escolta activa a l'API, procés que malgasta recursos i és molt menys eficient. Tot i això, les desavantatges d'usar *webhooks* són un augment de complexitat en la gestió i recepció d'aquests, i que s'han de tenir en compte certs factors al implementar comunicació a través d'ells. Principlament, tota aplicació que rebi o gestioni *webhooks* ha de ser asíncrona per evitar colls d'ampolla de múltiples peticions seguides o, fins i tot per determinades tasques, condicions de carrera. Així i tot, són millors que l'alternativa a usar *webhooks*, que seria deixar un procés de `Xatbot` escoltant activament a Telegram per fer-li la petició de les dades de l'usuari. Per a més informació, vegis la secció 8.3.1

Capítol 4

Treballs Relacionats

En aquesta secció detallarem l'històric del projecte LovelaceUB, explicant els dos treballs que van començar el projecte [6] [5]; les funcionalitats que implementaven cadascun, l'estat del codi d'aquests projectes quant a estructura i de disseny, i en quin punt comença el projecte actual.

4.1 Lovelace UB

Lovelace UB és un projecte d'agent conversacional educacional desenvolupat en dos anteriors treballs de final de grau[6] [5]. L'objectiu d'ambdós projectes era desenvolupar un agent conversacional de suport a l'aprenentatge en el context de l'assignatura Programació I del grau d'Enginyeria Informàtica de la Universitat de Barcelona, específicament per l'assignatura de Programació I. Es volia oferir als alumnes coneixement de les assignatures - sobre el pla docent, avaluacions, dates d'exàmens i temari d'aquestes - així com correctors automàtics (emprant tests unitaris i llibreries de testatge) de manera que les alumnes poguessin demanar exercicis i resoldre'ls a través de l'agent conversacional. Així doncs, l'alumnat tindria a l'abast una eina addicional per a conèixer més sobre el temari i realitzar exercicis de l'assignatura sense necessitar retroacció directa del docent.

En el treball de Fernández Fernández [6], es va implementar una agent conversacional usant AIML (Artificial Intelligence Mark-up Language)[1] per a definir la interacció conversacional, fent servir GitHub per emmagatzemar tant el coneixement de l'agent conversacional com els exercicis de l'assignatura, i amb interfície d'usuari de Telegram. L'aplicació utilitzava *webhooks* tant a Telegram com a GitHub per a rebre els missatges des de Telegram i contestar-los, així com per actualitzar el coneixement del bot quan s'actualitzava algun dels repositoris de les assignatures de GitHub. Continuant amb els exercicis, es

[1] AIML: llenguatge de marques basat en detecció de patrons per crear bots conversacionals

van habilitar les funcionalitats necessàries per enviar i rebre exercicis des de Telegram o a través d'un correu electrònic i també un mecanisme bàsic de correcció. La Figura 4.1 mostra el diagrama de programari del seu projecte.

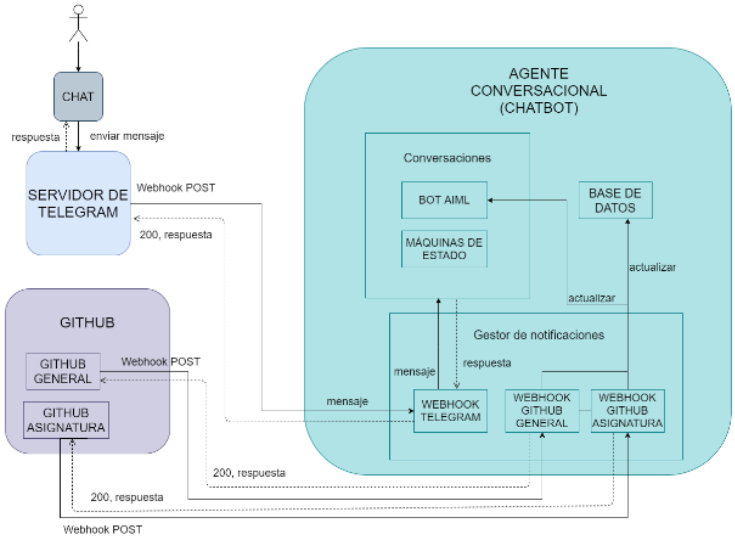


Figura 4.1: Diagrama de Software del projecte de Fernández

Al treball de Domènech Puig [5] es vàren implementar els correctors de Java i Python dins del bot fet per Fernández Fernández en dues imatges de Docker, una per cada llenguatge, per garantir una execució segura i evitar-ne d'externes en una potencial bretxa de seguretat. Els Dockers tenien un volum enllaçat amb l'aplicació principal que contenia tots els tests disponibles. Quan es demanava la correcció d'un exercici, es buscava el test pertinent a la carpeta compartida, s'executava dins del Docker i s'obtenien els resultats dels tests. La Figura 4.2 mostra el diagrama de programari del seu projecte.

4.2 Problemes i solucions

Al principi, l'objectiu d'aquest projecte era pujar l'aplicació desenvolupada per Doménech i Puig a una plataforma de serveis al núvol (Heroku), testejant profundament les funcionalitats dels correctors de Java i Python i implementant les millores necessàries al codi del seu projecte; un cop fet, es volia implementar el corrector de C seguint l'estructura dels altres dos correctors per l'assignatura de matemàtiques. Finalment, i a causa de problemes trobats de diversos tipus que s'expliquen a continuació, es va decidir implementar des de zero aquest projecte, no reutilitzant (practicament) cap part dels projectes anteriors.

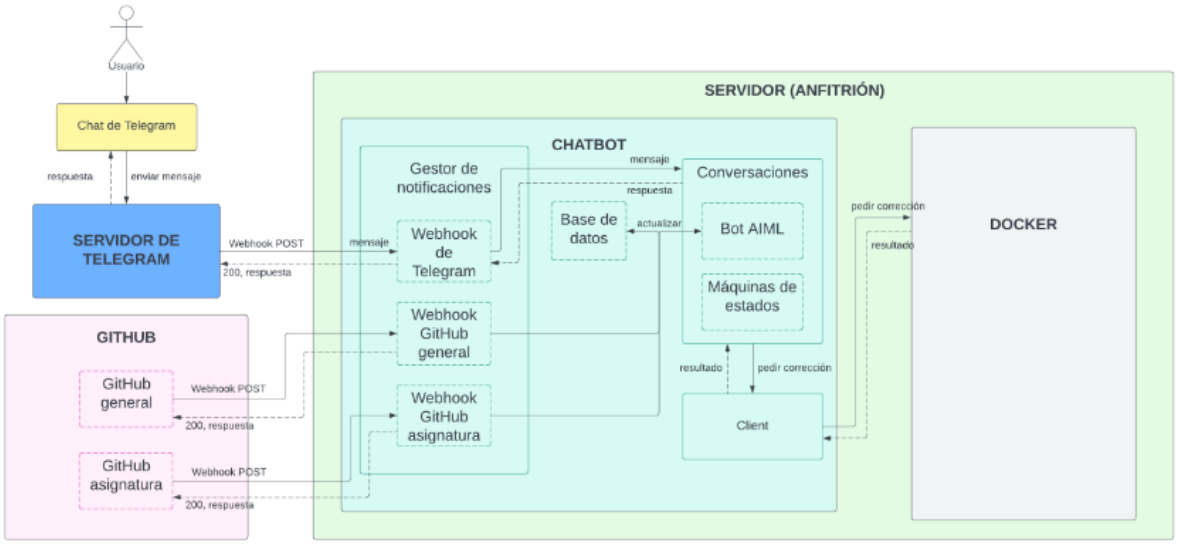


Figura 4.2: Diagrama de Software del projecte de Domenech

Problemes de Desplegament

El primer problema va ser amb la plataforma d'Heroku. Aquesta no permetia penjar una aplicació que empri un Docker sense penjar el contenidor ja construït o construint-la en el moment de pujada, i la implementació original del projecte no feia cap de les dues. Doménech i Puig encenia el Docker quan era requerit, executava els fitxers i tests pertinents i l'apagava, una molt bona pràctica que s'ha mantingut al codi final, ja que evita potencials riscos de seguretat (secció 8.7.2). Per pujar doncs l'aplicació a Heroku, es va decidir dividir-la en dues parts: la comunicació amb Telegram i la gestió dels correctors de Java i Python, i *dockeritzant*-ne les dues parts, pujar-les a Heroku. Això va requerir un canvi molt fonamental a l'estructura del projecte, ja que l'app principal *dockeritzada* es comunicaria amb els altres dos Dockers (correctors de Java i Python) mitjançant *sockets* (seguint la implementació de Domenech Puig) i compartiria els fitxers de testatge creant un volum per accedir a les correccions.

Problemes de Disseny

L'aproximació per a fer el desplegament dita al paràgraf anterior (penjar la imatge construïda directament o construir-la en el moment de pujada) no era possible implementar-la directament. El problema de *dockerizar* l'aplicació principal era que la carpeta contenint els tests era buida fins que l'aplicació no engegava i l'omplia. Com que el volum és el primer que es crea quan es construeix el contenidor, l'aplicació compartia la carpeta buida mitjançant el volum, i després la carpeta s'omplia, deixant els correctors amb una carpeta

buida sense accés als tests. Per tant, es va decidir tenir una base de dades externa, de manera que en realitzar el desplegament a qualsevol servei d'allotjament es pogués implementar de manera dedicada. En aquest punt ens vàrem començar a adonar que les dependències internes del codi existents dels altres dos projectes eren molt més grans del que es pensava, cosa que feia molt difícil extreure la base de dades, ja que tenia moltes crides al llarg de tot el codi.

Durant aquest procés, també ens vàrem adonar que el servidor implementat a càrrec de gestionar les comunicacions amb Telegram no implementava el *webhooks* asíncronament ni tampoc posava les estructures de control necessaries pel correcte funcionament de l'aplicació, vegis la secció 8.3.1, deixant gran part del codi prèviament escrit virtualment no reutilitzable, ja que els canvis requerits eren massa grans. Sumant-li a aquest fet que el codi tenia moltes dependències entre si, i que s'empraven un identificador temporal per la persistència de dades a la base de dades (s'feia servir el *chat_id*, que pot canviar per connexió i no el *user_id*, que és únic per cada usuari) cosa que portaria a un usuari ser incapaç d'accedir a les seves dades emmagatzemades prèviament, vàrem pensar que intentar reutilitzar-lo podria ser molt costós.

Solució

Com s'ha descrit als apartats anteriors, l'aplicació original tenia problemes tant de disseny com de desplegament. A més, els objectius principals d'aquest projecte requerien canvis significatius tant per la tecnologia de l'agent conversacional (d'AIML a RASA) com pel nou corrector per a llenguatge C. Per tant, es va decidir implementar aquest projecte des de zero al voltant d'aquests dos grans objectius. Així doncs, aquest projecte proposa un disseny modular, seguint bones pràctiques de programació, amb patrons de programari (com *Factory* o *Singleton* [28]), amb una base de dades més senzilla però independent i afegint-li tot el que sigui considerat necessari per al projecte, i també reutilitzant petites parts de disseny i codi dels dos anteriors projectes. A la Figura 7.1 de la secció 7.1, es pot veure l'estat final del disseny del projecte.

Capítol 5

Marc Conversacional Rasa: Característiques i Funcionament

Rasa és un terme ambigu que es pot referir a diferents entitats. El cor de Rasa és **Rasa Open Source**, conformada per la tecnologia i els models que proveeixen a l'agent conversacional, és una plataforma d'intel·ligència artificial de desenvolupament d'agents conversacionals proveïda per aprenentatge automàtic. Aquesta està sota la llicència *Apache 2.0* i és de codi obert, podent-se consultar a GitHub[15] tot el codi font.

Rasa també es refereix a l'empresa que continua desenvolupant i mantenint *Rasa Open Source* i que té dos productes que hi afegeixen funcionalitats, *Rasa Pro* i *Rasa Enterprise*. Aquests dos productes conformen el que s'anomena la *Rasa Platform*, tal com es pot veure a la Figura 5.1.

La *Rasa Platform* és la solució de l'empresa Rasa a oferir a empreses o particulars per desenvolupar agents conversacionals adaptats a les seves necessitats, amb *Rasa Pro* oferint diverses funcionalitats extres (com *End-to-End Testing* o visualització de dades d'entrenament avançades) i *Rasa X/Enterprise* que dona control a elements de més baix nivell del projecte pel tractament de dades més massiu.

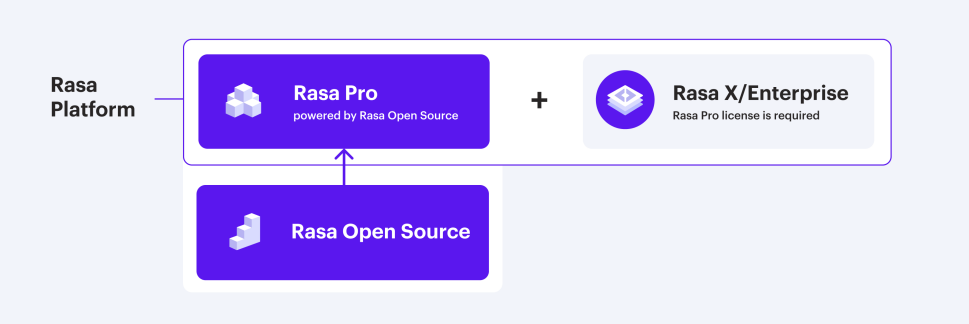


Figura 5.1: Estructura de la *Rasa Platform*, compresa de la tecnologia (*Rasa Open Source*) i dels productes, la *Rasa Platform*.

En aquest projecte s'ha emprat exclusivament *Rasa Open Source* i cada vegada que es mencioni Rasa, el text s'estarà referint exclusivament a *Rasa Open Source*.

Rasa és un marc conversacional per la construcció d'agents conversacionals que permeten entendre i mantenir converses en llenguatge natural i executar codi de tercers. Per fer-ho, Rasa utilitza tècniques d'aprenentatge automàtic amb dades proporcionades pel dissenyador i dels mateixos usuaris. Els conceptes principals de Rasa són els següents:

- Intenció (o *intent*): la intenció que l'usuari manifesta. Per exemple, una intenció de l'usuari pot ser saludar.
- Pronuncia (o *utter*): la pronúncia és l'acció que realitza el bot després de detectar una intenció, és a dir, què contesta.

Per norma general, es proveeix a Rasa amb exemples d'una intenció concreta de l'usuari que consten de diverses fases amb el mateix significat, tal com es pot veure a la Figura 5.2b. Després s'associa una pronúncia amb el mateix nom a aquesta intenció amb un o més texts de resposta, que és la que l'agent automàticament cridarà després de detectar la intenció de l'usuari. A la Figura 5.2a es pot veure un exemple de les dades d'entrenament del projecte.

```yaml
nlu:
- intent: greet
  examples: |
    - ei
    - hola
    - Hola!
    - que tal
    - com va
    - bon dia
    - bones
    - salut!
    - que passa
    - molt bones
    - bona tarda
```

(a) Dades d'entrenament de la intenció 'greet'

```yaml
- intent: know_when_exam
  examples: |
    - Quan és el [final](exam) de [progra](subject)?
    - Quin dia és el [parcial](exam) de [MENU1](subject)?
    - El [final](exam) de [PROG1](subject) quan serà?
    - Quan serà el [parcial](exam) de [menu1](subject)
    - En quin dia cau el [final](exam) de [metodes](subject)
    - [parcial](exam), lloc, [Mètodes Numèrics 1](subject)
    - [final](exam), data, [Programació 1](subject)
    - El [parcial](exam), de [programacio 1](subject) serà el 24/08/2023?
    - El [final](exam) de [programación](subject) quan és?
```

(b) Dades d'entrenament de la intenció 'know_when_exam' amb les entitats *exam* i *subject*

Figura 5.2: Exemples d'entrenament d'aprenentatge automàtic

Per extreure informació d'un missatge de l'usuari, Rasa empra les anomenades entitats (o *entities*), les quals són definides abans de l'execució. Aquestes entitats representen objectes que l'agent ha d'identificar en un missatge rebut i usar els valors que prenen per decidir la resposta que ha de retornar. Tot i això, les entitats són efímeres, és a dir, no es guarden a memòria més enllà d'un intercanvi de dos missatges entre la usuària i l'agent. Per dotar a l'Agent Conversacional amb memòria a llarg termini i poder mantenir el context de la conversa, s'usen ranures (o *slots)*. Aquestes ranures normalment s'omplen amb el valor d'una entitat del mateix nom; n'hi ha de diferents tipus i es poden omplir de diferents maneres, per exemple, es pot definir una ranura booleana en funció de si una intenció concreta de l'usuari és detectada per l'agent, posant la detecció d'aquesta a Cert i la no detecció a Fals.

L'extracció d'entitats es pot fer amb diverses tecnologies proveïdes per Rasa i que s'executen en un ordre determinat pel programador. Aquestes varien des de *Tokenizers* [1] a cercar patrons amb expressions regulars[2] fins a l'extracció amb el model *DIETClassifier*[13] (*Detector of Intents and Entities Transformer).* Aquest model és una xarxa neuronal amb arquitectura *transformer* que s'entrena amb dades proveïdes pel programador. Està dissenyada per detectar intencions de l'usuari i entitats contingudes en un missatge. A l'entrenament, el programador marca les entitats detectables en cada intenció, i si les dades són prou representatives, les extreu per a ser guardades a la memòria de l'agent o siguin utilitzades per les polítiques per predir-ne la següent acció.

```
- rule: Say 'I am a bot' anytime the user challenges
  steps:
  - intent: bot_challenge
  - action: utter_iamabot

- rule: Check the entities when asking about the date
  steps:
  - intent: know_when_exam
  - action: action_check_entities_when
  wait_for_user_input: false

- rule: Check the entities when asked about the place
  steps:
  - intent: know_where_exam
  - action: action_check_entities_where
  wait_for_user_input: false
```

(a) Tres exemples de les estructures de Rasa

```
- story: Place of an exam from a subject, exam not provided, subject provided
  steps:
  - intent: know_where_exam
  - action: action_check_entities_where
  - action: utter_ask_exam_type_where
  - intent: select_exam
  - action: utter_know_where_exam

- story: Place of an exam from a subject, subject not provided, exam provided
  steps:
  - intent: know_where_exam
  - action: action_check_entities_where
  - action: utter_ask_subject_where
  - intent: ask_subject
  - action: utter_know_where_exam
```

(b) Dos exemples de històries de Rasa

Figura 5.3: Exemples de regles i històries de l'agent

El funcionament de l'agent serà determinat per regles (o *rules*), que són conjunts d'instruccions que l'agent ha d'obeir. Les més senzilles enllacen intencions de l'usuari amb

[1] *Tokenizer*: Detector de seqüències comunes en un text, tradueix caràcters a *Tokens*

[2] Expressions regulars (o *regex*): seqüències de paraules que conformen un patró de cerca en un text.

pronuncies, per exemple, de la Figura 5.3a. Seguint la Figura, es pot veure com es defineix una intenció (*intent*: bot_challenge), que s'activa si l'usuari pregunta al bot si és realment un bot. El següent pas és l'acció *(action: utter_iamabot)* que ha de prendre l'agent just després de detectar la intenció. Per tant, aquesta regla diu que si una usuària pregunta si ets un *xatbot*, l'agent pronunciï que efectivament és un *xatbot*.

En cas de voler marcar fluxos més complexos a l'agent, també existeixen les històries (o *stories*), trobant-ne dos exemples a la Figura 5.3b.

Les accions (o *actions)* són classes de Python cridables per l'agent. Aquestes es poden posar en regles o en històries, fet que l'agent les accioni quan són activades. Des de les accions es poden fer peticions a altres servidors per obtenir informació externa o es pot fer crides a ell mateix per marcar-li una o múltiples accions a realitzar quan l'acció acabi d'executar.

Per acabar, tenim els formularis (o *forms*), que combinen tots els conceptes que s'han mencionat fins ara. Un formulari és definit marcant una sèrie de ranures (*slots*) que necessiten prendre valor per poder dur a terme una acció posterior, també definida a priori. El formulari s'activa a través d'una regla que connecta una intenció amb un bucle, que es trencarà quan les ranures hagin pres totes valor, i fins que no tinguin el valor, l'agent les continuarà preguntant a l'usuari. Aquesta eina és molt efectiva per assegurar-se que totes les ranures tenen valors comprovats i controlats abans de fer peticions a l'exterior de l'aplicació. En aquest projecte els s'han fet servir per assegurar-se que, en demanar un enunciat de l'exercici, tots els valors (dificultat, tipus i assignatura) no estiguin absents. Per més detall, vegis la secció 7.3

Tots aquests conceptes es relacionen amb el següent flux de treball simplificat que es mostra a la Figura 5.4, que ensenya els passos generals de com es genera una resposta. Quan el sistema rep un missatge de l'usuari, l'*interpreter* l'analitza (1), cercant les entitats i intencions que pot portar aquest missatge. Tal com s'ha mencionat anteriorment en aquest apartat, aquest és un dels dos punts que Rasa pot emprar aprenentatge automàtic amb el *DIETClassifier*. Fent-ne cinc cèntims, el *DIET* prediu amb una probabilitat quina és la intenció que és més probable que la usuària tingués al enviar aquest missatge i també n'extreu entitats si se'n podien extreure. Un cop l'agent ha escollit la intenció de probabilitat més alta, es passa al següent punt, el *Tracker* (2).

El *Tracker* és un registre de la conversa que l'agent conversacional ha tingut amb l'usuari fins al moment. No només dona accés al text, sinó que et permet accedir a intencions detectades a missatges anteriors, accions predites pel model o les ranures i els seus valors. Un cop s'han detectat la intenció i les entitats, el *tracker* es guarda el missatge, la intenció, les entitats i omple les ranures detectades.

El següent pas per l'agent és decidir la *Policy* [19] a complir (3), que és l'encarregat de predir com ha de contestar l'Agent. Hi ha diverses polítiques que Rasa implementa per predir la resposta; les més comunes són la *RuleBasedPolicy* (política a partir de regles) la

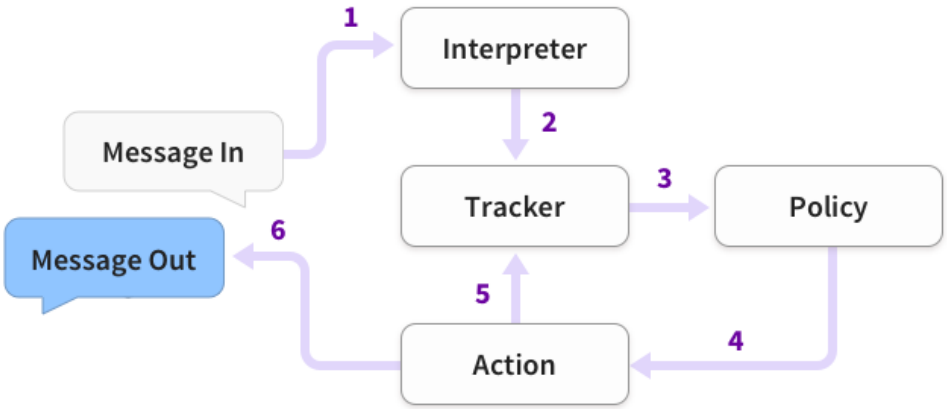


Figura 5.4: Arquitectura del sistema Rasa [18]

qual cerca si hi ha una regla prèviament definida aplicable en aquest cas concret. També trobem les polítiques que empren aprenentatge automàtic, com la TEDPolicy (*Transformer Embedding Dialogue*). La TED implementa també arquitectures *transformers* per donar una probabilitat a predir quina acció ha de prendre l'agent a continuació.

Un cop l'acció ha estat decidida (4), aquesta es registra al *tracker* com a acció presa (5) i el missatge s'envia a l'usuari (6).

Capítol 6

Anàlisi

Aquest capítol especifica les funcionalitats implementades al projecte que faciliten la interacció usuari-agent. L'usuari és un alumne de la Facultat de Matemàtiques i Informàtica el qual pot estar cursant Programació 1 del grau d'Enginyeria Informàtica, Mètodes Numèrics 1 o 2 del grau de Matemàtiques o ambdues si és estudiant de la doble titulació.

L'usuari empra la interfície de Telegram per comunicar-se amb l'agent. Les intencions de l'usuari poden ser:

- Preguntar dubtes sobre el pla docent d'una de les dues assignatures. Concretament, quan o on són els exàmens finals o parcials.
- Preguntar dubtes conceptuals i/o tècnics sobre els continguts de l'assignatura.
- Registrar-se per poder sol·licitar exercicis al bot conversacional i corregir-los, que són les funcions següents.
- Demanar enunciats d'exercicis a implementar que varien depenent de l'assignatura de la qual estigui parlant l'agent amb l'usuari. Aquests enunciats disposen també de tipus (quin tema de l'assignatura tracten) i de dificultat (varia depenent l'assignatura) i l'usuari els pot escollir específicament o deixar-los buits perquè sigui aleatòri. Els exercicis de Programació 1 estan fets amb el llenguatge Java, i els de Mètodes Numèrics amb el llenguatge C.
- Enviar la correcció d'un exercici per comprovar si passa tots els tests. Un cop corregits pel sistema mitjançant tests unitaris, s'envia la retroacció dels tests superats juntament amb una nota del 0 al 10.

L'última funcionalitat que implementa l'agent és l'actualització dels exercicis quan el docent de l'assignatura els canviï, ja que automàticament rebrà una notificació que farà que actualitzi totes les dades locals, però aquesta no és interactuable per l'alumne.

La Figura 6.1 descriu els casos d'ús en el qual una usuària pot interactuar amb l'Agent Conversacional:

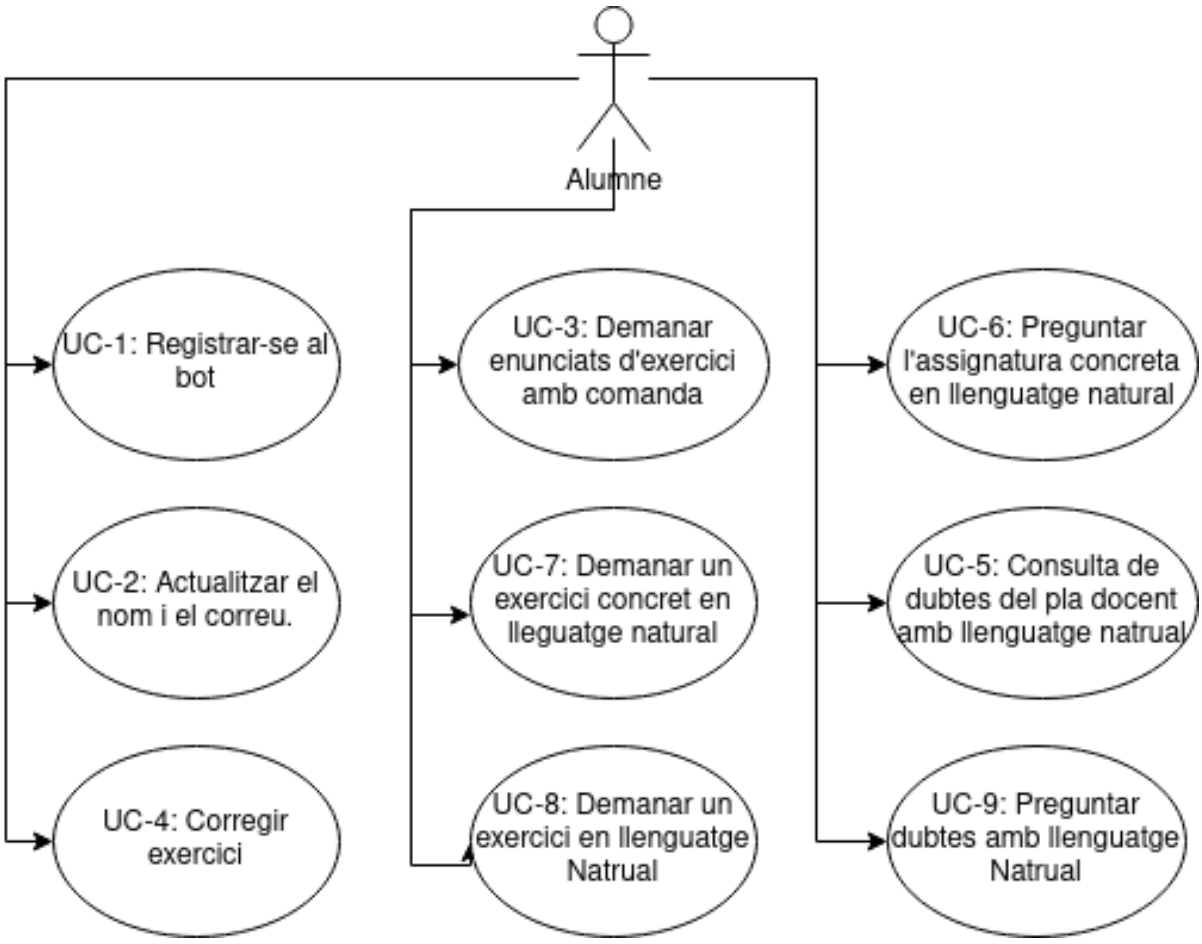


Figura 6.1: Diagrama de casos d'ús

Nom	UC-1 Registrar-se a l'agent conversacional
Rol	Alumne
Descripció	L'alumne es registra a l'agent conversacional per accedir a les funcionalitats de demanar exercicis i correcció d'aquests. Es registra donant el seu nom i un correu electrònic.
Condicions prèvies	La usuària no s'ha registrat
Flux normal	1. La usuària usa la comanda `/registra <nom> <correu>` 2. L'agent conversacional envia un missatge de confirmació referint-se a l'usuari pel seu nom.
Flux alternatiu	1. (a) L'agent conversacional respon que algun argument no és correcte 2. (a) L'agent conversacional respon que hi ha hagut un error a la base de dades
Condicions Posteriors	L'alumne està guardat dins del sistema

Taula 6.1: Cas d'ús 1: Registra-se al bot

Nom	UC-2 Actualitzar el nom i el correu
Rol	Alumne
Descripció	L'alumne vol canviar les seves dades.
Condicions prèvies	L'usuari s'ha registrat
Flux normal	1. L'usuari usa la comanda `/registra` 2. L'agent conversacional envia un missatge confirmant que les credencials han estat actualitzades.
Flux alternatiu	1. (a) L'agent conversacional respon que el nombre d'arguments no és correcte 2. (a) L'agent conversacional respon que hi ha hagut un error a la base de dades
Condicions Posteriors	L'alumne ha actualitzat les seves dades

Taula 6.2: Cas d'ús 2: Actualitzar el nom i el correu

El registre al sistema que tracten la UC-1 i la UC-2, no és un registre convencional. El que s'emmagatzema al sistema, i que serveix com a identificació inequívoca de l'usuari, és la *user_id* de Telegram, un paràmetre que Telegram assigna a cada usuari. L'agent l'extreu i l'usa com a identificació inequívoca de cada alumne que usa la comanda registra. Això també significa que un usuari no pot tenir múltiples perfils, ja que l'identificador és la id, no el nom i el correu. Tot i així, independentment del registre o no, el bot guarda els *logs* de qualsevol usuari, registrat o no.

Nom	UC-3 Demanar enunciat d'exercici amb comanda
Rol	Alumne
Descripció	L'alumne vol demanar a l'agent conversacional un enunciat d'un exercici per resoldre'l
Condicions prèvies	La usuària s'ha registrat al sistema
Flux normal	1. La usuària usa la comanda `/exercici` amb els arguments: (a) `<assignatura>` si el vol aleatòriament (b) `<assignatura> <tipus> <dificultat>` si el vol aleatòriament amb un tipus i dificultat concrets (c) `<assignatura> <nom_fitxer>` si vol l'enunciat de l'exercici nom_fitxer 2. L'agent conversacional envia l'enunciat de l'exercici. Si l'exercici ha estat intentat abans, envia un altre missatge comunicant-li a l'usuari. Si no ha estat exitós, l'agent conversacional envia un missatge explicant l'error.
Flux alternatiu	1. (a) L'agent conversacional contesta que no és una assignatura correcta (b) L'agent conversacional contesta que no reconeix el tipus i la dificultat proporcionats per l'alumne. (c) L'agent conversacional contesta que no ha trobat el fitxer. 2. (a) L'agent conversacional envia 'Com que la teva nota és menor que un 7, t'he enviat aquest exercici altre cop' si l'usuari ja havia intentat aquest exercici, però tenia menys d'un 7 de nota. Si té més d'un 7 no enviarà els exercicis. (b) L'agent conversacional envia un missatge avisant que hi ha hagut un problema buscant els tests.

Condicions Posteriors	L'alumne ha obtingut un enunciat a resoldre.

Taula 6.3: Cas d'ús 3: Demanar enunciat d'exercici

Nom	UC-4 Corregir exercici
Rol	Alumne
Descripció	L'alumne vol corregir la seva implementació de l'enunciat
Condicions prèvies	La usuaria està registrada i té un exercici a corregir
Flux normal	1. L'usuari adjunta la seva correcció a Telegram i l'envia escrivint l'assignatura corresponent a la llegenda. 2. L'agent conversacional respon que comença la correcció. 3. L'agent conversacional envia els resultats de la correcció (tests superats, fallats i nota). 4. L'agent conversacional envia, segons sigui el cas: (a) Si l'exercici no havia estat intentat mai per l'usuari, un missatge informant que s'ha guardat la correcció nova. (b) Si la nota de l'intent actual és superior a l'emmagatzemada, un missatge dient que l'usuari ha superat la seva anterior nota i aquesta s'actualitza. (c) Si la nota de l'intent actual és inferior a l'emmagatzemada, un missatge a l'usuari dient que no ha superat la seva anterior nota.

Flux alternatiu	1. (a) L'usuari no omple la llegenda 2. (a) L'agent conversacional cerca si la usuària ha mencionat una assignatura prèviament. Si la troba, cerca l'exercici (b) L'agent conversacional respon que no troba el test associat a aquest nom d'exercici a la base de dades, tant si el nom del fitxer o l'assignatura no es troben. (c) L'agent conversacional envia que hi ha hagut un error o un *time out* a la correcció. 3. L'agent conversacional envia que hi ha un problema amb la base de dades i que no guardarà els resultats.
Condicions Posteriors	L'alumne ha obtingut els resultats de la seva implementació juntament amb una nota numèrica.

Taula 6.4: Cas d'ús 4: Corregir exercici

Nom	UC-5 Consulta de dubtes sobre el pla docent amb llenguatge natural
Rol	Alumne
Descripció	L'alumne té dubtes sobre quan és algun examen d'alguna assignatura
Condicions prèvies	Cap
Flux normal	1. L'usuari pregunta quan/on és l'examen d'una assignatura i: (a) menciona el tipus d'examen (parcial/final) i l'assignatura (programacio1, metodes). (b) només menciona el tipus d'examen. (c) només menciona l'assignatura. 2. L'agent conversacional detecta la intenció de l'usuari de saber quan/on és un examen d'un tipus 3. L'agent conversacional detecta al missatge les entitats: (a) El tipus d'examen i l'assignatura concreta (b) El tipus d'examen (c) L'assignatura concreta 4. En funció de l'opció 3, l'agent conversacional: (a) pronuncia quan/on és el tipus d'examen de l'assignatura concreta. (b) pronuncia la pregunta 'Quin tipus d'examen?' i mostra botons per la selecció 'Parcial' o 'Final' (c) pronuncia la pregunta 'De quina assignatura' sense mostrar botons. 5. L'usuari (b) escull Parcial o Final a través dels botons (c) escriu l'assignatura. 6. L'agent conversacional pronuncia la data/lloc del tipus d'examen d'una assignatura concreta.

Nom	UC-5 Consulta de dubtes sobre el pla docent amb llenguatge natural
Rol	Alumne
Flux alternatiu	2. L'agent conversacional no detecta la intenció de l'usuari i realitza una acció imprevista o no en detecta cap i pronuncia l'acció per defecte 'No t'he entès, ho podries repetir si us plau?'. 3. (a) L'agent conversacional no detecta cap de les entitats previstes. (b) L'agent conversacional detecta una entitat no prevista (ni *exam* ni *subject*, però sí un altre tipus) 4. (a) L'agent pronúncia que l'usuari ha de mencionar l'assignatura i el tipus d'examen. (b) L'agent segueix el flux previst i pronunciarà 'No t'he acabat d'entendre :(' el missatge per defecte de la intenció preguntar quan o on és un examen.
Condicions Posteriors	L'usuari sap quan/on és un tipus d'examen d'una assignatura concreta.

Taula 6.5: Cas d'ús 5: Consulta de dubtes sobre el pla docent

Nom	UC-6 Preguntar l'assignatura actual amb llenguatge natural
Rol	Alumne
Descripció	L'alumne vol saber de quina assignatura és l'última que l'agent conversacional té constància.
Condicions prèvies	Cap
Flux normal	1. L'usuari pregunta sobre quina assignatura han parlat anteriorment l'agent i ell. 2. L'agent conversacional detecta la intenció de l'usuari de saber l'assignatura. 3. L'agent conversacional envia l'assignatura en una frase.
Flux alternatiu	1. L'agent conversacional detecta que no s'ha parlat de cap assignatura i retorna el missatge "No m'has mencionat cap assignatura encara".
Condicions Posteriors	L'alumne sap l'assignatura en discussió

Taula 6.6: Cas d'ús 6: Preguntar l'assignatura actual

Nom	UC-7 Demanar exercici concret llenguatge natural
Rol	Alumne
Descripció	L'alumne vol tornar a rebre l'enunciat d'un exercici perquè ha esborrat l'enunciat o l'ha perdut
Condicions prèvies	L'alumne sap el nom de l'exercici que demanarà
Flux normal	1. L'alumne demana al bot que l'hi enviï un exercici mencionant: (a) el nom de l'exercici i l'assignatura (b) només el nom de l'exercici 2. L'agent conversacional detecta la intenció de l'usuari de voler un exercici en contret. 3. L'agent conversacional, segons el cas del pas 1: (a) Pronuncia què procedeix a buscar el fitxer, anar al punt 6 (b) Pronuncia de quina assignatura és el fitxer mencionat 4. L'usuari contesta de quina assignatura és. 5. L'agent conversacional detecta l'assignatura. 6. L'agent conversacional envia l'exercici a l'usuari.
Flux alternatiu	1. (a) L'alumne no menciona el nom, però sí l'assignatura, aleshores és el cas d'ús 7 (UC-8 Taula 6.8) 2. (a) L'agent conversacional no detecta la intenció de l'usuari i realitza una acció imprevista o no en detecta cap i pronuncia l'acció per defecte. 6. L'agent conversacional pronuncia que no ha trobat l'exercici de l'assignatura.

Condicions Posteriors	L'alumne rep l'enunciat de l'exercici proporcionat.

Taula 6.7: Cas d'ús 7: Demanar exercici concret amb llenguatge natural

Nom	UC-8 Demanar un exercici llenguatge natural
Rol	Alumne
Descripció	L'alumne vol un exercici escollint el tipus i la dificultat
Condicions prèvies	Cap
Flux normal	1. L'alumne demana un exercici: (a) mencionant l'assignatura. (b) no mencionant l'assignatura. 2. L'agent conversacional detecta la intenció de voler un exercici: (a) amb l'assignatura, saltar al punt 5. (b) sense l'assignatura. 3. L'agent conversacional pronuncia a l'usuari de quina assignatura vol l'exercici. 4. L'alumne escriu l'assignatura. 5. L'agent conversacional pregunta a l'usuari mitjançant botons quina és la dificultat de la qual vol l'exercici 6. L'alumne respon escollint una dificultat de les disponibles 7. L'agent conversacional pregunta a l'usuari mitjançant botons de quin tipus vol l'exercici. 8. L'alumne respon escollint un tipus dels disponibles 9. L'agent conversacional pronuncia que ha rebut les dades correctament. 10. L'agent conversacional envia el fitxer.

Nom	UC-8 Demanar un exercici llenguatge natural
Rol	Alumne
Flux alternatiu	1. L'agent conversacional no detecta la intenció de l'usuari i realitza una acció imprevista o no en detecta cap i pronuncia l'acció per defecte. 8. L'agent conversacional no ha entès correctament l'entitat assignatura i pronuncia un missatge comunicant-ho a l'usuari, 9. L'agent conversacional respon que: (a) no hi ha exercicis disponibles amb aquest tipus o dificultat. (b) hi ha hagut un problema recuperant l'exercici.
Condicions Posteriors	L'alumne sap l'assignatura en discussió.

Taula 6.8: Cas d'ús 8: Demanar exercici aleatori amb llenguatge natural

Nom	UC-9 Preguntar dubtes generals
Rol	Alumne
Descripció	L'alumne vol resoldre un dubte general de l'assignatura
Condicions prèvies	Cap
Flux normal	1. La usuària escriu 'Tinc un dubte:' seguit del dubte que té. 2. L'agent conversacional detecta la intenció de l'usuari de resoldre un dubte. 3. L'agent conversacional genera la resposta al dubte de l'usuari
Flux alternatiu	2. (a) L'agent conversacional no detecta la intenció i crida a l'acció per defecte (a) L'agent conversacional detecta un dubte invàlid, i respon que no pot ser resolt.
Condicions Posteriors	L'alumne ha respost el dubte

Taula 6.9: Cas d'ús 9: Preguntar dubtes generals.

Capítol 7

Disseny

Aquest capítol representa el disseny de l'aplicació: els mòduls que consta i quines funcionalitats s'encarreguen. Després es discuteix el disseny de l'agent conversacional implementant tant per les comandes de Telegram com per Rasa i juntament amb el disseny de la llibreria de testatge matemàtic amb CUnit.

7.1 Diagrama de Programari

L'aplicació consta de tres mòduls: `Xatbot`, `GestorDB` i `Corrector`, cadascun amb el seu propi servidor i un únic propòsit. La Figura 7.1 mostra el diagrama de programari amb les relacions entre els tres. La decisió de partir l'aplicació en tres parts és semàntica en naturalesa i responent al principi de responsabilitat única[29]: el mòdul de `Xatbot` gestiona la recepció i resposta de peticions des de Telegram, mentre que `GestorDB` s'ocupa de mantenir les dades actualitzades rebent les actualitzacions des de GitHub i gestionar les peticions dels altres mòduls. Per acabar, el `Corrector` rep els fitxers d'exercicis a corregir i encén els Dockers de correcció. Tot i en els següents apartats tractar-se a fons què fa i com està implementat cada mòdul, primer llistem en profunditat les quines funcionalitats implementa cada un:

- `Xatbot`: Rep els missatges de l'usuari a través d'un *webhook* amb l'aplicació que proporciona el la interfície gràfica. Aquest és el mòdul principal de l'aplicació que implementa totes les funcionalitats relacionades amb l'usuari:
  - Permet a l'usuari registrar-se per accedir a les correccions d'exercicis i als enunciats.
  - Envia els enunciats dels exercicis als alumnes.

  - Gestiona els sistemes de correcció quan els alumnes envien un exercici a corregir.
  - Implementa la lògica de l'agent conversacional: entén les intencions dels usuaris (NLU) i respon tan amb respostes predefinides amb l'ús de Rasa com amb respostes generades (NLG) quan s'usa *GPT-3.5-turbo*.

  Aquestes funcionalitats es poden realitzar en comandes (vegis la secció 8.1) o bé mitjançant una conversa en llenguatge natural (vegis secció 7.3)

- `GestorDB`: Gestiona totes les dades de l'aplicació amb una base de dades (vegis MariaDB a la secció 8.1):
  - Emmagatzema tota la informació de l'aplicació: correccions realitzedes, correus i identificadors dels usuaris, enunciats i tests d'exercicis.
  - Descarrega tots els exercicis de cada assignatura des del repositori de Github.
  - Reflexa els canvis dels repositoris a la base de dades mitjançant una petició *webhook*.
  - Aten les peticions dels altres mòduls que involucren dades: el `Corrector` demana tests i les dades dels usuaris o enunciats d'exercicis des de `Xatbot`.

- `Corrector`: Crea, gestiona i obté resultats dels contenidors de Docker que corregeixen les implementacions dels alumnes. S'activa al rebre una petició de correcció de `Xatbot` i demana els tests a `GestorDB`, executa i envia els resultats a `Xatbot`.

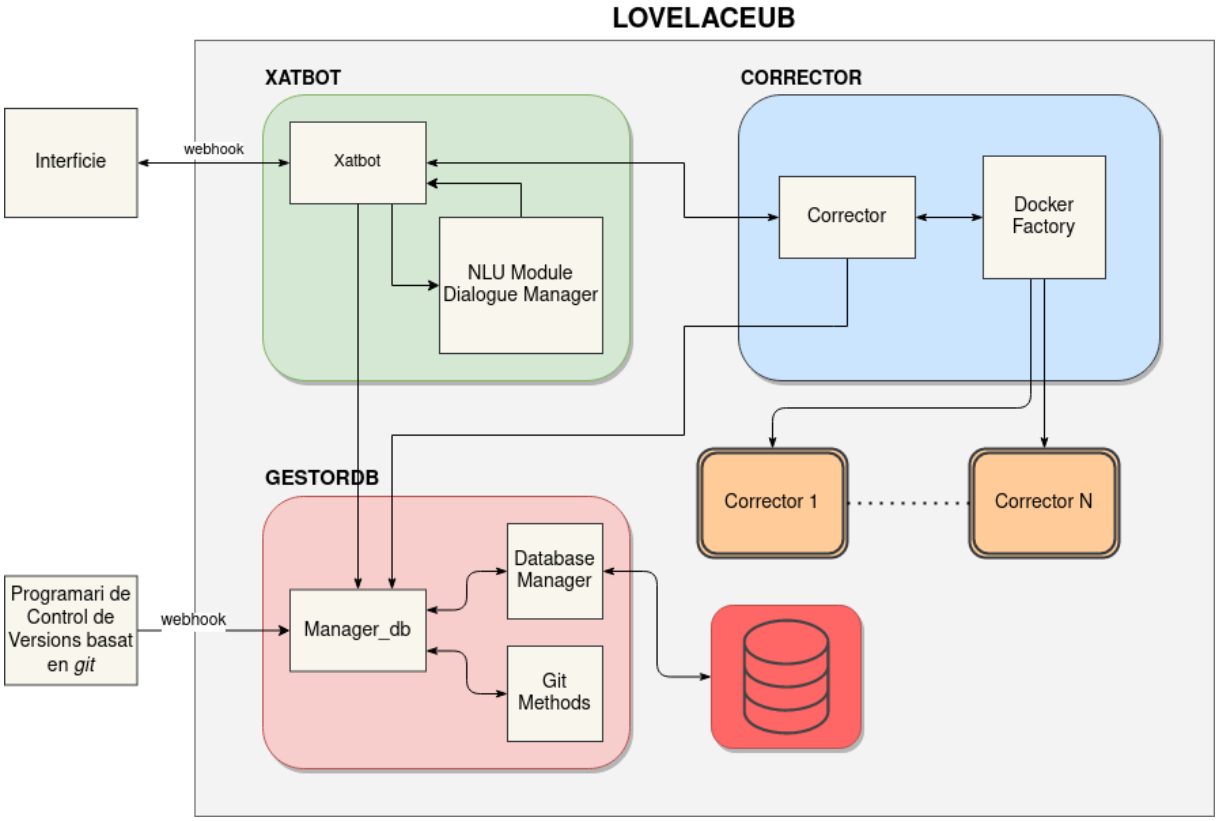


Figura 7.1: Diagrama de Programari del Projecte

Cal notar que els mòduls taronjes amb doble línia són contenidors de Docker. D'ara endavant, ens referirem als mòduls del projecte com a `Xatbot`, `GestorDB` i `Corrector`.

7.2 Xatbot i Telegram

El mòdul `Xatbot` està format per dues parts diferenciades, la part de comunicació amb Telegram amb l'arquitectura i la part de *NLP* proveïda per Rasa. La Figura 7.2 es mostra el mòdul `Xatbot` en detall. Aquest mòdul és un servidor web FastAPI construït sobre un exemple de *python-telegram-bot*[14] per gestionar *webhooks* amb una cua prioritària per evitar potencials problemes d'execució (vegis secció 8.3.1), per tant, és l'aplicació que gestiona els missatges de l'usuari un cop arriben.

En essència, quan Telegam activa el *webhook*, `Xatbot` pot realitzar diverses accions. Si el missatge conté una comanda de Telegram (`/comanda`, vegis la secció 8.1) es crida la funció associada corresponent. Si el missatge no és una comanda, es crida el submòdul de Rasa per escollir una resposta apropiada. Per diversos passos dins de les comandes o

de Rasa, el `Xatbot` té disponible respostes predefinides en un JSON (vegeu secció 7.2.4) per encadenar diversos passos amb els següents o donar retroacció a certs esdeveniment.

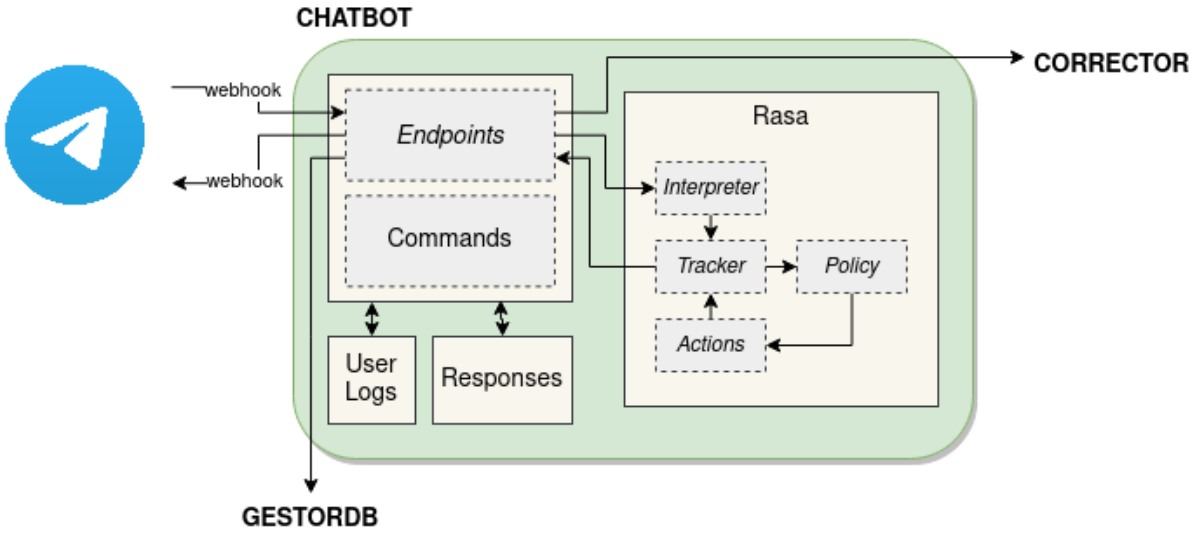


Figura 7.2: Diagrama de software en detall del mòdul `Xatbot`

Qualsevol de les tres rutes de missatges mencionades al paràgraf anterior poden utilitzar els *endpoints* (vegis apèndix A.5) de `Xatbot` per demanar a `GestorDB` dades de l'aplicació necessàries per a l'execució.

7.2.1 Comandes

Telegram és la interfície escollida per comunicar-se amb el bot conversacional. Tant la interfície com la llibreria de Python emprada pel projecte faciliten al programador l'ús de comandes (de l'estil `/comanda [arguments]`) a les que se'ls hi pot associar un comportament en concret.

Les comandes proporcionen una manera eficient de comunicar-se amb l'agent si l'usuari sap quins són els valors que poden prendre els arguments, ja que no s'ha implementat una funcionalitat d'autocompletat a Telegram mentre l'usuari els escriu. Per tant, són una opció pensada per usuaris més experimentats amb l'entorn i les seves capacitats. Totes les comandes implementades estan relacionades amb un dels casos d'ús dels descrits a la secció 6, i es mencionarà quin específicament a mesura que apareguin.

- `/registra` : guarda l'usuari a la base de dades, donant-li accés a les correccions i a demanar exercicis al bot. (UC-1: Taula 6.1) Si l'usuari ja estava registrat, tornar a fer aquesta comanda actualitzarà les seves dades per les noves que proporcioni (UC-2: Taula 6.2)

- `/exercici`: retorna un exercici perquè l'alumne n'implementi la solució (UC-3: Taula 6.3). Permet:
  - `/exercici <assignatura> <nom_fitxer>`: envia el fitxer de l'assignatura amb el nom concret, per si l'usuari ha esborrat l'enunciat sense voler, ha perdut el fitxer original o vol tornar a començar de nou.
  - `/exercici <assignatura>`: envia a l'usuari un exercici aleatòriament de l'assignatura.
  - `/exercici <assignatura> <tipus> <difficultat>`: envia un exercici aleatori pertanyent a l'assignatura indicada, amb la dificultat i el tipus concrets.
- Correcció d'exercici (UC-4: Taula 6.4) Per corregir un exercici que l'usuari ja ha implementat només l'ha d'enviar a Telegram juntament amb l'assignatura a la llegenda, tal com es mostra a la Figura 7.3. No és possible corregir l'exercici si l'usuari no escriu l'assignatura

Figura 7.3: Enviament d'una solució posant l'assignatura a la llegenda

7.2.2 Comanda `/registra`

Aquesta comanda implementa la UC-1 i UC-2 (Taules 6.1 6.2). La funció registra espera dos arguments, el nom i el correu de l'usuari. Si l'usuari no estava registrat, s'emmagatzema a la base de dades, però si ho estava, s'actualitzen les seves dades a la base de dades.

Enviar la comanda sense arguments amb l'usuari no registrat al bot activa una resposta predefinida de l'agent informant a l'usuari com s'usa la comanda. En canvi, si l'usuari sí està registrat, enviarà un missatge saludant a l'usuari amb el seu nom.

A la Figura 7.4 s'hi troba un diagrama de seqüència de la funció.

7.2.3 Comanda `/exercici`

Aquesta comanda implementa la UC-3 (Taula 6.3). La funció exercici canvia el comportament radicalment segons el nombre d'arguments.

Si no hi ha cap argument, l'agent contesta un missatge predefinit amb les instruccions d'ús de la comanda. Si hi ha un argument, la funció assumeix que és una assignatura i envia a l'usuari un exercici aleatori independentment del tipus i de la dificultat. Si hi ha dos arguments, la funció assumeix que el primer és l'assignatura i el segon el nom del fitxer concret que l'usuari vol que l'agent li enviï. Si hi ha tres arguments, la funció assumeix que van en l'ordre assignatura, tipus i dificultat i envia un exercici aleatori coincidint amb els paràmetres de l'usuari.

A la Figura 7.5 s'hi troba el diagrama de flux concret de l'explicació superior.

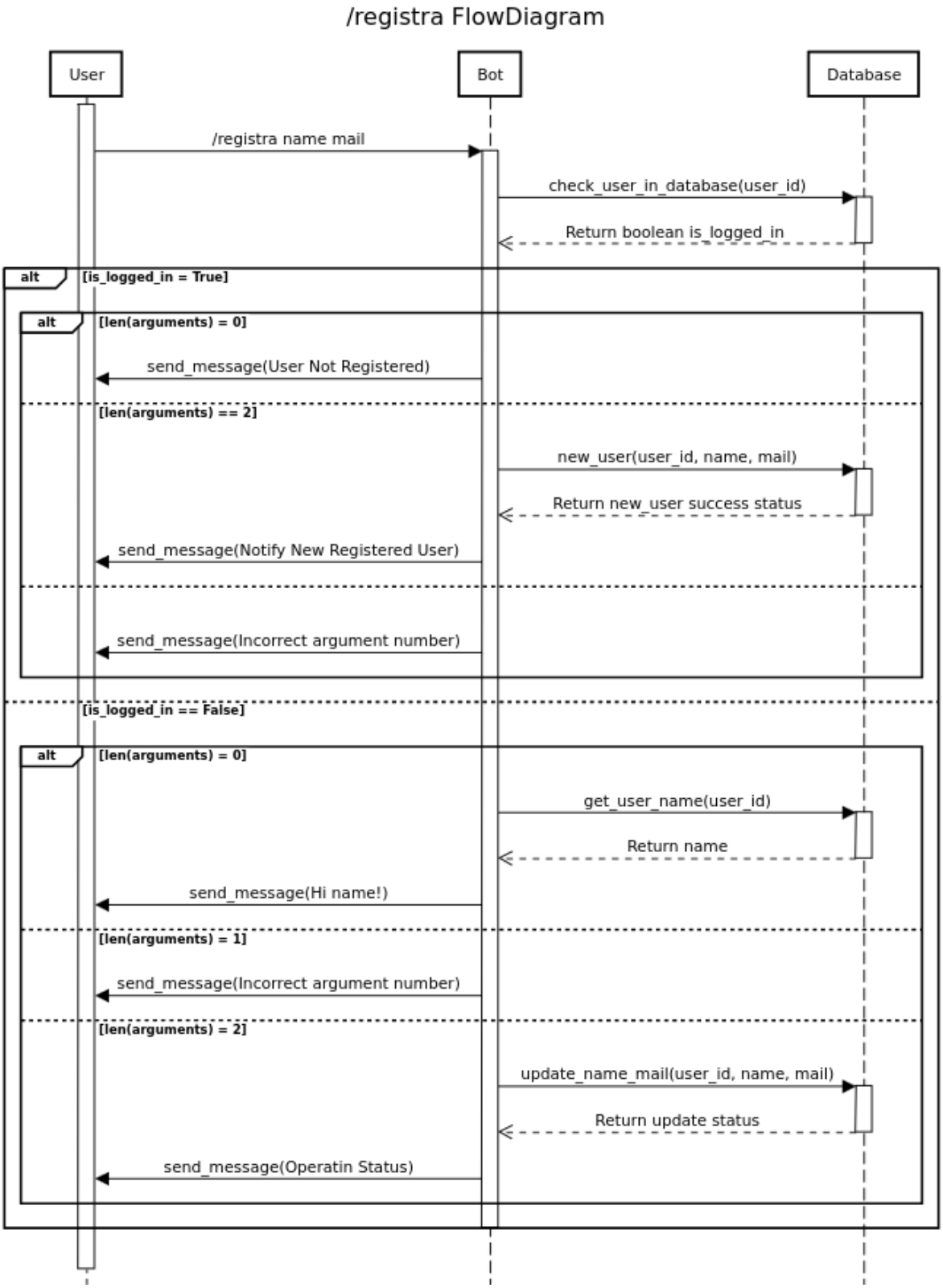


Figura 7.4: Diagrama de seqüència de la funció associada a /registra

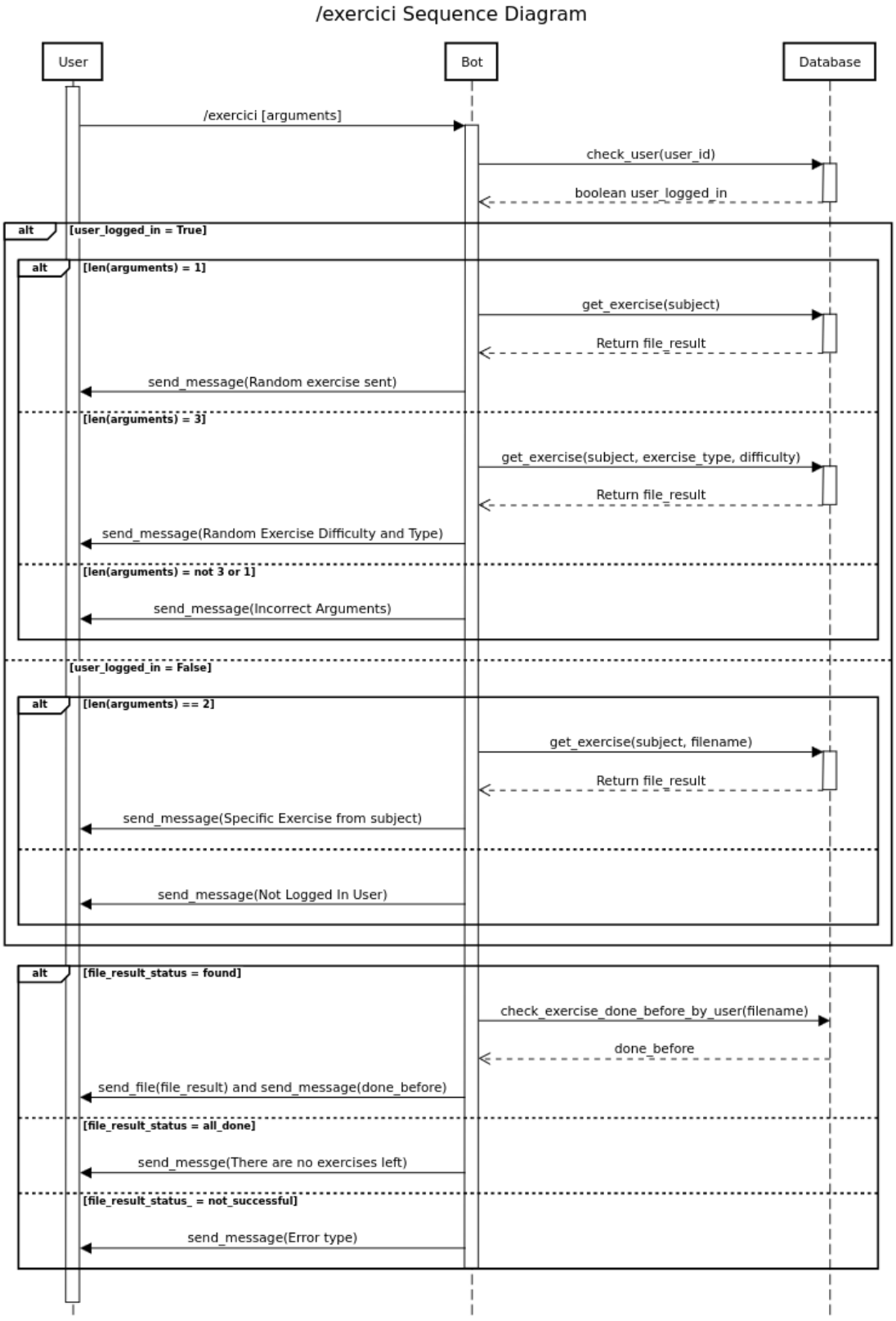


Figura 7.5: Diagrama de seqüència de la funció associada a /exercici

7.2.4 Respostes Predefinides

A certes accions de l'usuari, l'Agent Conversacional ha de respondre sempre de la mateixa manera, com per exemple: el missatge de començar una correcció, el missatge de què no s'ha trobat un fitxer, missatges d'error sobre arguments de les comandes o sobre problemes ocasionats per altres mòduls. Aquests missatges no són usats pels mòduls d'NLP de Rasa, sinó per controlar diversos errors o missatges de les comandes, com respostes per donar retroacció o els resultats dels tests.

Totes aquestes respostes es troben en dos fitxers JSON, els quals s'accedeixen a diversos punts del codi per després enviar-se a l'usuari. A la Figura 7.6 a continuació es poden veure exemples de quins missatges s'han implementat.

```
{
    "place_holder": "He rebut un missatge de text!",
    "signup" : {
        "loggedin_noargs" : "Ja et conec a tu {name}!",
        "loggedin_noargs_notfound" : "No t'he trobat a la base de dades, em
        "updated" : "He actualitzat el teu nom i correu amb el que m'has don
        "non_signup" : "Sembla que no t'has registrat. Si vols fer-ho, escri
        "bad_argument" : "Ep! Necessito el nom i el correu",
        "success" : "Perfecte! Benvingut {name}, no t'oblidaré :)",
        "error" : "He tingut un problema afegin-te :( Si us plau, prova-ho m
    },
    "file_handler" : {
        "error" : "No he trobat el fitxer. Assegura't que el nom és correcte
    },
    "placeholder_audio" : "M'ha arribat un audio",
    "exercici" : {
        "specific": "Buscant l'exercici {}, un moment si us plau...",
        "no_args" : "No he entès bé els arguments que m'has donat... Deixa'm
        "not_found" : "L'exercici que m'has dit no existeix :(",
        "not_found_arguments" : "Sembla que el tipus {} i dificultat {} no e
        "no_entra" : "Si vols que t'envii un exercici aleatòri hauries de en
        "error" : "Perdona, he tingut un problema trobant l'exercici. Si us
        "already_done": "Com que la teva nota és menor que un 7, t'he enviat
        "signed_up_error" : "He tingut un problema buscant-te com a usuari.
        "random_exercise": "T'envio un exercici aleatòri!",
        "random_parameters" : "T'envio un exercici aleatòri de la dificultat
        "all_done": "Uau, realment has aconseguit fer tots els exercicis? Et
        "exit": "Aqui el tens, molta sort :)",
        "user_not_found": "No t'he trobat a la base de dades, segur que t'ha
    },
```

Figura 7.6: Respostes predefinides de l'agent conversacional

Aquesta aproximació a guardar els missatges per separat i no dins del codi és una bona pràctica que incorpora molts avantatges, com la facilitat de canviar l'idioma del bot només canviant quin fitxer es carrega al principi de l'execució, canviar missatges fàcilment només coneixent-ne les claus d'entrada, cercar el missatge que es vol canviar directament al JSON.

7.3 Rasa

En aquesta secció volem adreçar com s'ha dissenyat l'agent conversacional i quines característiques de Rasa hem fet servir per cada funcionalitat. En general, trobarem equivalències d'aquests subapartats per cada cas d'ús descrit a la secció 6 que involucrin llenguatge natural.

7.3.1 Conversa 1: Quan/On és un Examen d'una Assignatura

Les converses sobre la data o el lloc estan dissenyats amb els mateixos passos i seguint funcions iguals en concepte, per tant, en el següent concepte només s'emprarà el *Quan* tot i ser equivalent amb la conversa *On*.

S'ha definit una intenció per l'usuari *know_when_exam* que conté dues entitats: *exam* i *subject*. Si aquestes es detecten al missatge de l'usuari, s'emmagatzemen a la ranura que té el seu mateix nom.

```
- intent: know_when_exam
  examples: |
    - Quan és el [final](exam) de [progra](subject)?
    - Quin dia és el [parcial](exam) de [MENU1](subject)?
    - El [final](exam) de [PROG1](subject) quan serà?
    - Quan serà el [parcial](exam) de [menu1](subject)
    - En quin dia cau el [final](exam) de [metodes](subject)
    - [parcial](exam), lloc, [Mètodes Numèrics 1](subject)
    - [final](exam), data, [Programació 1](subject)
    - El [parcial](exam), de [programacio 1](subject) serà el 24/08/2023?
    - El [final](exam) de [programación](subject) quan és?

- intent: know_where_exam
  examples: |
    - On és el [final](exam) de [progra](subject)?
    - A quina aula és el [parcial](exam) de [MENU1](subject)?
    - A quina aula serà el [final](exam) de [PROG1](subject)?
    - Es a l'aula B7 el [parcial](exam) de [programacio](subject)
    - Em podries dir l'aula del [final](exam) de [metodes](subject)
    - [Programació 1](subject), lloc, [parcial](exam)
    - A on es fa el [final](exam) de [Mètodes Numèrics 1](subject)
    - A on es farà el [parcial](exam) de [programación]
```

Figura 7.7: Dades d'entrenament de la intenció 'quan és un examen de l'assignatura'

Tal com es veu a la Figura 7.7, la paraula entre claudàtors és el valor de l'entitat i la paraula entre parèntesis és el nom de l'entitat que s'ha d'extreure. El *DIETClassifier* entrenarà amb aquestes dades, i un cop les detecti al missatge de l'usuari, les guardarà

a la seva ranura corresponent. L'algoritme 1 descriu el procés que segueix la conversa Quan/On és un examen d'una assignatura assumint que totes les intencions i entitats als missatges de l'usuari es prediuen correctament.

```
- rule: Check the entities when asking about the date
  steps:
  - intent: know_when_exam
  - action: action_check_entities_when
  wait_for_user_input: false

- rule: Check the entities when asked about the place
  steps:
  - intent: know_where_exam
  - action: action_check_entities_where
  wait_for_user_input: false
```

Figura 7.8: Regla que activa la comprovació de les entitats de la intenció 'quan és un examen de l'assignatura'

Després, mitjançant la regla que es veu a la Figura 7.8 es crida a l'acció *action_check_entities_when*, que comprova el valor de les entitats. Si totes dues apareixien al missatge original, dins de l'acció *check_entities* (que es troba al mòdul d'Accions de Rasa en un fitxer Python) es crida l'acció *utter_know_exam_when*, pronunciant la resposta associada a la intenció detectada. Per aquesta raó, no es veu a la regla de la Figura 7.8 un '*utter_know_exam_when*, ja que es crida dins del codi Python associat a l'acció. Com que l'acció a més és la que decideix el següent pas de l'agent, s'ha d'informar a la regla que no esperi un missatge de l'usuari automàticament, que és el *wait_for_user_input* al final de les dues regles.

Cal dir per la naturalesa de l'aprenentatge automàtic, si no es detectés l'entitat correctament, hi ha polítiques de Rasa que permeten predir quina és l'acció més probable que l'agent hagi de prendre. Aquesta és la *Memoization*, tal com es pot veure a '*else* de l'algoritme.

Si una de les dues entitats no ha pres valor, l'agent aplicarà una de les dues històries de la Figura 7.9, la que fa que el bot pregunti a l'usuari l'entitat que li falta i després l'extregui. La Figura 7.10 mostra una possible conversa entre l'usuari i l'agent.

```
- story: Date for an exam from a subject, exam not provided, subject provided
  steps:
  - intent: know_when_exam
  - action: action_check_entities_when
  - action: utter_ask_exam_type_when
  - intent: select_exam
  - action: utter_know_when_exam

- story: Date for an exam from a subject, subject not provided, exam provided
  steps:
  - intent: know_when_exam
  - action: action_check_entities_when
  - action: utter_ask_subject_when
  - intent: ask_subject
  - action: utter_know_when_exam

- story: Date for an exam from a subject, exam and subject not provided
  steps:
  - intent: know_when_exam
  - action: action_check_entities_when
  - action: utter_know_when_exam
```

Figura 7.9: Histories que regeixen la política *TED* per la predicció d'accions de la intenció 'quan és un examen de l'assignatura'

Algorithm 1 Passos de l'algoritme associat a la UC-6: saber quan/on és l'examen d'una assignatura.

Require: Intent: know_when_exam trained
Require: Entities. exam: Text, subject: Text.
Require: Slots: exam.filled_by(exam: entity), subject.filled_by(subject: entity).
Ensure: input.

```
intent, exam, subject = DIETClassifier.extract(input)
if intent = know_when_exam then
    action_check_entities = rule.next_action(intent)
    FollowupAction(action_check_entities())
    exam, subject = tracker.slot(exam, subject)
    if exam = None then
        new_input = tracker.FollowupAction(utter_ask_exam_buttons)
        exam = user_response(DIET.extract(new_input))
    end if
    if subject = None then
        new_input = tracker.FollowupAction(utter_ask_subject)
        subject = user_response(DIET.extract(new_input))
    end if
    dispatcher.utter(utter_know_when_exam)
else
    next_action = Memoization.predict(intent)
end if
```

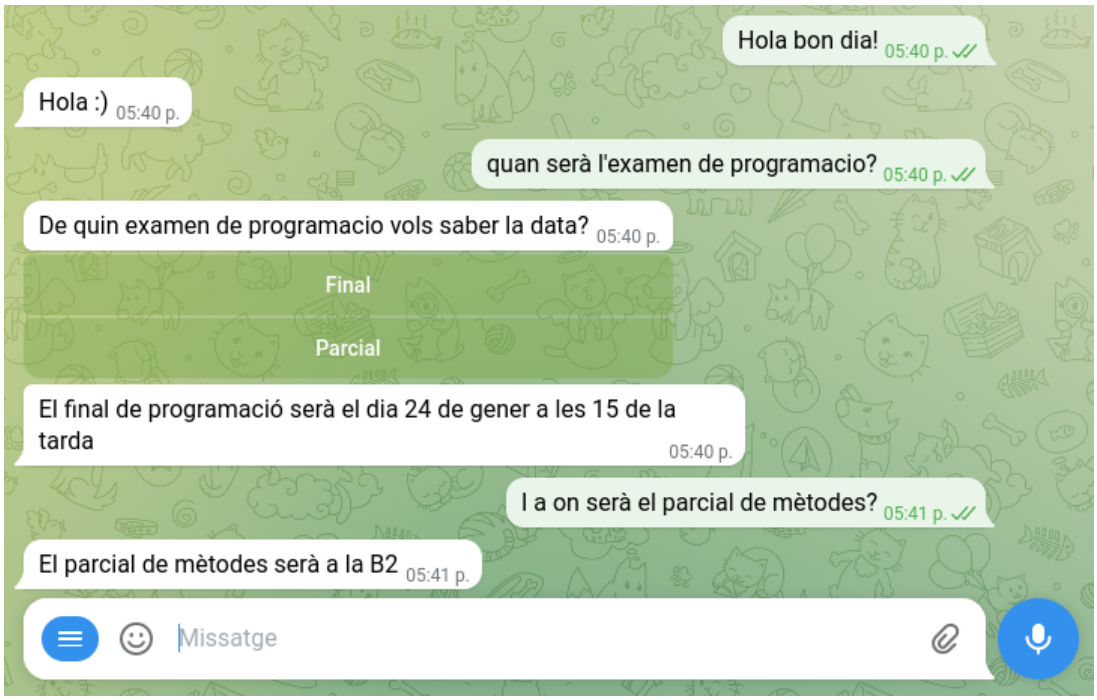


Figura 7.10: Exemple d'una conversa on l'usuari pregunta quan és l'examen de programació i on és el parcial de mètodes.

7.3.2 Conversa 2: Preguntar Assignatura

Aquesta interacció amb l'agent és molt directe. El que volem és que, a petició de l'usuari, el bot retorni el valor de la ranura assignatura (UC-5: Taula 6.5). Aquesta informació és rellevant, ja que les ranures són la memòria de l'agent mentre la sessió amb l'usuari està activa, i això pot afectar a comportaments d'altres UC.

Per exemple, considerem la UC-5 (Taula 6.5), preguntar on és el final de programació. Suposem que l'usuari ha escrit un missatge mencionant 'examen final' i 'programació' i que l'agent ha detectat la intenció, les dues entitats (*exam* i *subject*) i les ha carregat a les ranures. Aleshores, a la pregunta 'Quan és l'examen parcial' de l'usuari just després de la resposta, l'agent no preguntarà de quina assignatura, sinó que dirà quan és el final de programació. Això és perquè el valor de la ranura *subject* segueix sent programació, i per tant l'acció *check_entities_when* la detectarà amb un valor vàlid. El valor de la ranura es mantindrà a programació fins que l'usuari tanqui la connexió amb l'agent o fins que l'agent detecti un altre valor diferent per la ranura *subject*.

Tot i així, la possibilitat d'esborrar les ranures un cop Rasa les ha emprat mitjançant una acció és possible, però per disseny no s'ha pres, ja que dona pas a una interacció més natural amb l'agent. Aquesta naturalitat (que l'agent recordi detalls, nombres, entitats etc) de missatges anteriors de la conversa s'anomena la capacitat de l'agent de mantenir el

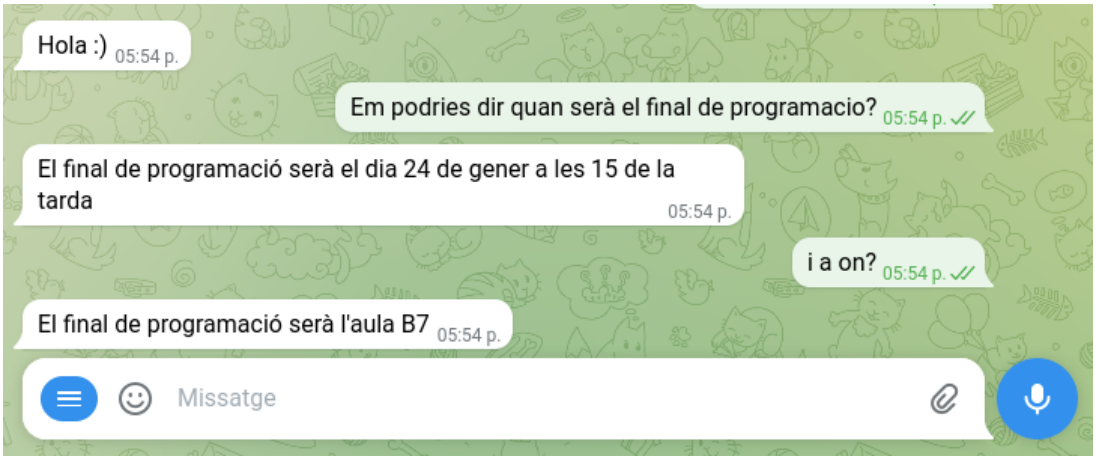


Figura 7.11: Exemple de conversa usuari-agent on l'agent és capaç de mantenir el context de la conversa i contestar adequadament la pregunta 'on és el final de programació' sense necessitat de l'usuari de repetir ni 'final' ni 'programacio'

context de la conversa, i mantenir els valors de les ranures en aquest concret ajuda molt. Continuant amb l'exemple del paràgraf anterior, assumir que l'usuari continuarà parlant de la mateixa assignatura diversos missatges amb l'agent un cop l'ha mencionat és un comportament humà natural; tanmateix, si la volgués canviar, també seria natural que li mencionés a l'interlocutor que la conversa passarà a tractar una altra assignatura. Tal com es mostra a la Figura 7.11 aquesta decisió de disseny permet que aquestes converses més emergents succeeixin.

Tornant a la intenció preguntar assignatura es troba a la Figura 7.12 els exemples d'entrenament per l'agent. Si l'usuari envia un missatge amb la intenció de preguntar assignatura, el classificador *DIETClassifier* trobarà l'entitat *subject*.

```
- intent: know_subject
  examples: |
    - De quina assignatura estem parlant?
    - Quina assignatura tens a memòria?
    - Assignatura?
    - Quina assignatura estem parlant?
    - L'assignatura de la que estem parlant és...
    - Quina assignatura t'he mencionat?
```

Figura 7.12: Dades d'entrenament de la intenció preguntar assignatura. Es pot veure entre claudàtors el valor de l'entitat i entre parèntesi quina entitat és.

Tal com es veu a la Figura 7.13, quan es detecta la intenció es crida l'acció *action_check_subject*, que comprova el valor de la ranura. Si aquest valor és nul, s'envia un missatge al respecte, i si té un valor, s'envia un missatge dient l'assignatura a l'usuari.

```yaml
- rule: Say the currect subject on discussion if the user asks
  steps:
  - intent: know_subject
  - action: action_fetch_subject
```

Figura 7.13: Regla d'activació de la intenció 'preguntar assignatura'

Tal com la conversa anterior, l'acció *action_fetch_subject* s'encarrega de pronunciar l'assignatura de la qual l'usuari i l'agent han estat parlant o dient que no n'havien parlat de cap, i per tant la regla s'acaba sense necessitat d'especificar res més. A la Figura 7.14 es veu com sense context l'agent no sap l'assignatura, però després de preguntar-li sobre una assignatura sap contestar-li.

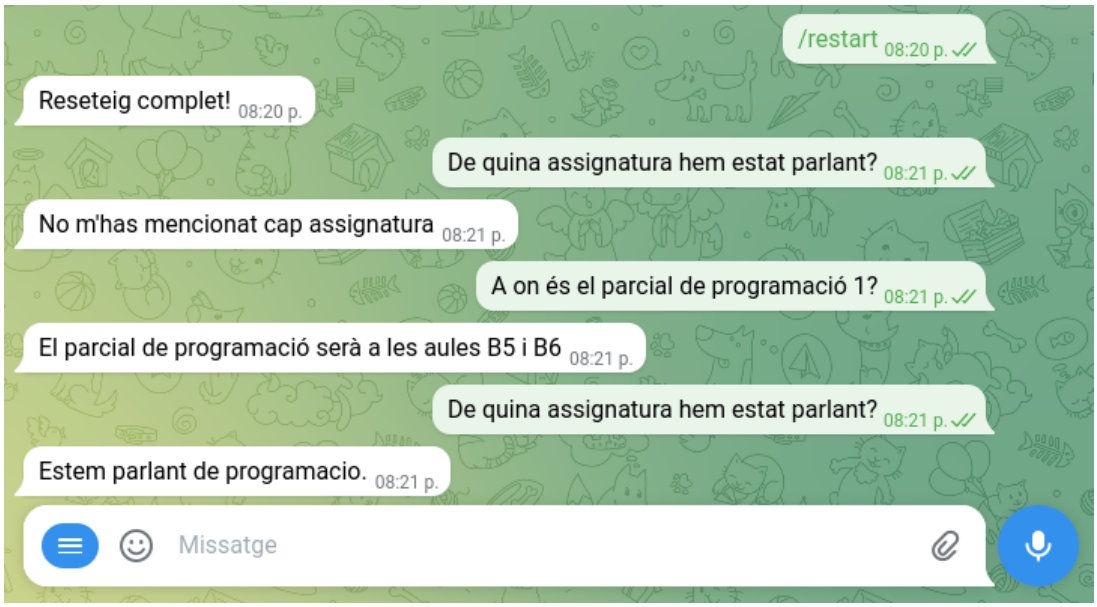


Figura 7.14: Conversa usuari-agent que ensenya les sortides de preguntar assignatura

7.3.3 Conversa 3: Demanar un Exercici Aleatori

Tot i poder-se implementar de manera molt semblant com s'ha implementat la intenció saber on/quan és un examen (vegis la secció 7.3.1) mitjançant accions personalitzades, s'ha optat per fer-ho amb un formulari. A la secció 7.3.7 s'expliquen les avantatges i inconvenients entre l'un i l'altre. Obviant-se detalls interns del tractament dels formularis de Rasa, l'algorisme 2 que regeix el seu funcionalment és el següent:

Algorithm 2 Passos de l'algoritme associat a la UC-6: Demanar un enunciat d'un exercici

Require: Intent: exercise_form
Require: Entities. subject: Text, difficulty: Text, exercise_type: Text.
Require: Slots: subject.filled_by(subject: entity), difficulty.filled_by(difficulty: entity), exercise_type.filled_by(exercsie_type: entity)
Require: Form(subject, difficulty, exercise_type)
Ensure: input

```
intent, [entities] = DIETClassifier.extract(input)
if intent = request_exercise then
    exercise_form = rule.next_action(intent)
    while subject ≠ None ∧ difficulty ≠ None ∧ exercise_type ≠ None do
        entities = DIET.extract(new_input)
        if subject ≠ None then
            dispatcher.utter(utter_ask_subject)
        end if
        if difficulty ≠ None then
            dispatcher.utter(utter_ask_difficulty)
        end if
        if exercise_type ≠ None then
            dispatcher.utter(utter_ask_exercise_type)
        end if
        next_action = rule.next_action(_) = exercise_form
    end whileFollowupAction(action_send_exercise)
end if
```

Tal com es veu a la Figura 7.15, un Formulari es defineix com les ranures (la memòria de l'agent) necessàries que han d'estar plenes. En el nostre cas, volem saber el valor de *subject*, *difficulty* i el *exercise_type*.

```
forms:
  exercise_form:
    required_slots:
      - subject
      - difficulty
      - exercise_type
```

Figura 7.15: Definició del formulari per la intenció 'Demanar Exercici'

La desactivació d'aquest formulari es gestiona internament des de Rasa, però la idea és que es quedarà en bucle mentre alguna de les ranures definides al formulari no tingui valor. Quan totes tres tinguin valor, la següent regla serà aplicable, i aquesta trencarà el formulari.

```
- rule: Activate Exercise Form
  steps:
  - intent: request_exercise
  - action: exercise_form
  - active_loop: exercise_form
```

(a) Regla de l'activació del formulari mitjançant l'acció '*exercise_form*'

```
- rule: Submit Exercise Form
  condition:
  - active_loop: exercise_form
  steps:
  - action: exercise_form
  - active_loop: null
  - slot_was_set:
    - requested_slot: null
  - action: action_send_exercise
```

(b) Regla de desactivació del formulari i de l'enviament de les dades recollides a través de l'acció '*action_send_exercise*'

Figura 7.16: Regles de gestió amb els formularis

La regla per entregar (*Sumbit*, Figura 7.16b) el formulari comprova si el bucle *active_loop* està activat amb el formulari. Si s'activa el formulari, però el bucle està apagat, significa que s'ha completat el formulari, i que, per tant, s'han de gestionar els resultats cridant l'acció *action_send_exercise*, que fa una crida a través de `Xatbot` fins a `GestorDB` per a enviar el fitxer a l'usuari. A la Figura 7.17 es mostra la conversa resultant. Cal matisar que l'usuari ha de registrar-se a l'agent primer, tal com es mostra a la Figura.

Figura 7.17: Exemple de conversa on es mostra com l'usuari es registra i seguidament demana un exercici, amb l'agent oferint botons per la selecció de la dificultat i el tipus.

7.3.4 Conversa 4: Demanar un Exercici Concret

Aquesta conversa és l'equivalent a la UC-7 (Taula 6.7) i correspon a demanar-li un exercici directament a l'agent quan l'usuari ja l'havia demanat abans o ja en sabia el nom. Separar la intenció *request_exercise* (la UC-8) de la *request_specific_exercise* (UC-7) rau en els següents dos motius:

- Tenir dues intencions amb exemples tan semblants podia dur a l'aprenentatge automàtic a confusió, a menys que hi hagués un gran nombre de dades d'usuaris per entrenar, que és el cas de la intenció *request_specficic_exercise* i les de *request_exercise*.
- Pel que fa a semàntica, és de fàcil justificació que per demanar un exercici aleatori és diferent a voler l'enunciat que l'usuari ja havia demanat, per tant, és coherent dividir-les en dos.

El disseny de l'algoritme empra el mateix funcionament amb accions personalitzades que comproven el valor de les entitats com la conversa on/quan és un examen d'una assignatura: quan el *DIETClassifier* detecta la intenció, hi ha una regla que crida a l'acció *action_check_entities_exercise*. Si a les ranures *filename* i *subject* hi ha valors, crida a l'acció *action_send_exercise*, definició que es troba en històries. En canvi, si la ranura de l'assignatura és buida, pronuncia preguntar-li a l'usuari l'assignatura. A la Figura 7.18 es mostra una interacció per demanar l'exercici concret amb el valor de la ranura *subject* a programació.

Figura 7.18: Exemple de conversa de l'usuari demanant un exercici concret a l'agent.

7.3.5 Conversa 5: Corregir Exercici

Aquesta secció correspon a la UC-4 (Taula 6.4). Consisteix en que l'usuari envia a l'agent el fitxer implementant la seva solució, i el *xatbot* el corregeix i n'envia retroacció. Aixi i tot, és requerit per l'usuari que ompli la llegenda amb l'assignatura, tal com hem ensenyat a la Figura 7.3 a menys que l'usuari hagi mencionat una assignatura a l'agent prèviament. És a dir, si l'usuari deixa la llegenda buida, es poden donar dos casos segons el valor de la ranura:

1. Si l'usuari mencionat alguna assignatura prèviament amb les Converses 1, 2 o 4 (vegis respectivament seccions 7.3.1, 7.3.3, 7.3.4), s'usarà el valor de la ranura *subject* si aquesta no està buida per fer la petició

2. Si l'usuari no ha mencionat cap assignatura, s'enviarà la petició amb un *None*, i per tant, la petició fallarà.

7.3.6 Conversa 6: Dubtes amb *GPT-3.5-turbo*

Aquesta secció correspon a la UC-9 (Taula 6.9). Aquesta és l'única conversa que implementa generació de text (*NLG*) i ho fa mitjançant una crida al model *GPT3.5-turbo*

d'OpenAI. El disseny de la intenció és senzill, tal com es pot veure a la Figura 7.19. Només consta d'una regla i d'una acció, que és la que executa la crida a l'API d'OpenAI.

```
- rule: Answer the doubt if there is a doubt
  steps:
  - intent: general_doubt
  - action: action_answer_general_doubt
```

Figura 7.19: Regla que connecta la intenció

La intenció *general_doubt* té la particularitat que està entrenada per detectar frases que comencin per 'Tinc un dubte:' i després el dubte en qüestió, per exemple: 'Tinc un dubte: què és un mètode numèric?'. Tal com s'ensenya a la Figura 7.20, la frase ha de començar per 'Tinc un dubte:' no per cap sinònim o succedani.

```
- intent: general_doubt
  examples: |
    - Tinc un dubte: em podries dir què és un for?
    - Tinc un dubte: com declaro una variable?
    - Tinc un dubte: quantes pomes em fan falta per apendre
    - Tinc un dubte: cap cap peu piu sense niu
    - Tinc un dubte: Hola pau, diga'm com estàs
    - Tinc un dubte: Ave, Maria, quan seràs meva?
    - Tinc un dubte: tres tristos tirgres menjen sègol
```

Figura 7.20: Dades d'entrenament de la intenció *general_doubt*

L'acció, com ja hem dit al paràgraf superior, fa la petició a l'API *GPT-3.5-turbo* per generar la resposta, la rep i la retorna a Telegram perquè la posi a pantalla. El que concretament s'envia al model és una petició (o *prompt*) escrita en llenguatge natural i el dubte de l'estudiant. La petició fixa concretament el text següent:

'Ets la Lovelace, un agent conversacional de suport a l'aprenentatge dels graus de Matemàtiques i Informàtica de la Universitat de Barcelona. Contesta en Català. La teva funció és resoldre els dubtes dels alumnes. Aquests començaran amb 'Tinc un dubte i els has de contestar en UN (1) paràgraf, és igual si s'allarga. Contesta els dubtes generals dels alumnes, però no contestis a peticions de resoldre dubtes específics o que impliquin implementar codi Exemples de dubtes contestables: què és Java?, com declaro una variable?, em podries explicar la seqüència de Fibonacci? Què és un mètode numèric i per a què serveixen? Què és el mètode de la potència? No contestis a dubtes de l'estil: implementa'm una funció que faci... En què falla aquest codi? Completa la frase/funció... Implementa el mètode de la potència... És a dir, no generis codi per l'alumne, només resoldre dubtes Empra to correcte i cordial però proper amb l'estudiant.'

Aquest és el resultat de fer *prompt engineering* o Enginyeria de Peticions[1] no molt extensiu

[1] *Prompt Enginyering:* desenvolupa, refina i optimitza peticions a models d'intel·ligències artificials

a causa de la limitació del temps inherent al marc d'aquest projecte. Les modificacions al petició més important realitzades de la original han estat Tot i això, aquesta petició ofereix resultat extremadament satisfactori. Mitjançant la perífrasi 'Tinc un dubte' s'ha aconseguit fer que l'agent:

- Es presenti, digui el seu nom i les seves capacitats.
- Respongui preguntes precises de caràcter tècnic
- Evitar que implementi exercicis per part dels alumnes, explicant el que hauria de fer la funció però no fent-ho.

La limitació més evident d'aquest sistema és que, al contrari que el sistema Rasa i l'*NLU*, la generació de contingut no és contextual: no té en compte els altres missatges que s'han enviat, només té en compte exclusivament el missatge actual. També, pel mateix disseny i per dificultar-ne l'abús, s'ha limitat a respostes exclusivament d'un paràgraf d'allargada. Tot i això, per la naturalesa d'aquests sistemes, és evident que s'haurà de refinar la petició. Tanmateix, es veurien avantatges computacionals (i pecuniaris, ja que el servei a l'API d'OpenAI no és gratuït) en reduir el nombre de *tokens* de cada consulta.

Cal destacar que la interacció amb el model *GPT-3.5-turbo* de manera controlada ofereix grans beneficis a l'Agent conversacional Lovelace i gran potencial de personalització i creixement, com implementar diferents accions que en facin ús, canviar les pronuncies estàtiques per una crida i generar totes les respostes o fins i tot demanar-li crear certs mini exercicis en temps real. Així i tot, pel disseny de l'agent no recomanaríem perdre el disseny d'intencions per passar a un de més general, ja que seria perdre l'essència del disseny i substituir-la per senzillament *ChatGPT*.

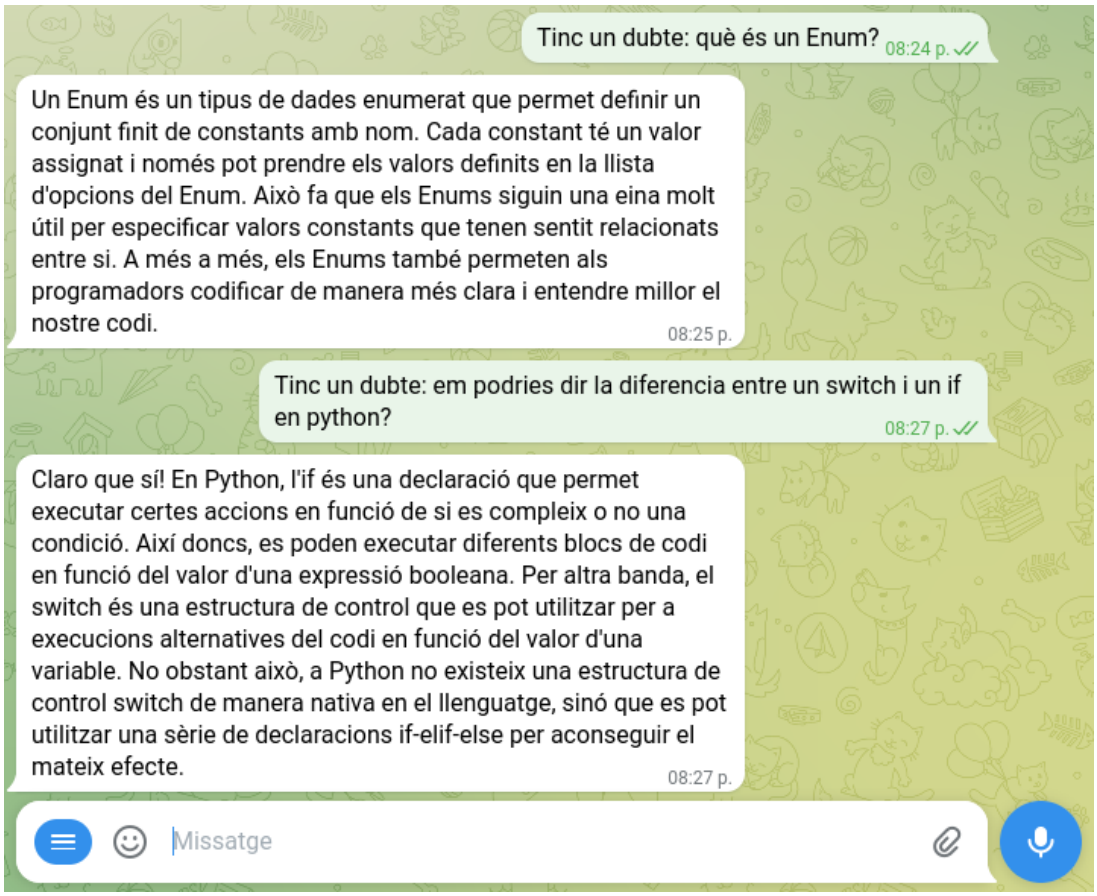


Figura 7.21: Respostes a preguntes tècniques generades pel model *GPT-3.5-turbo*

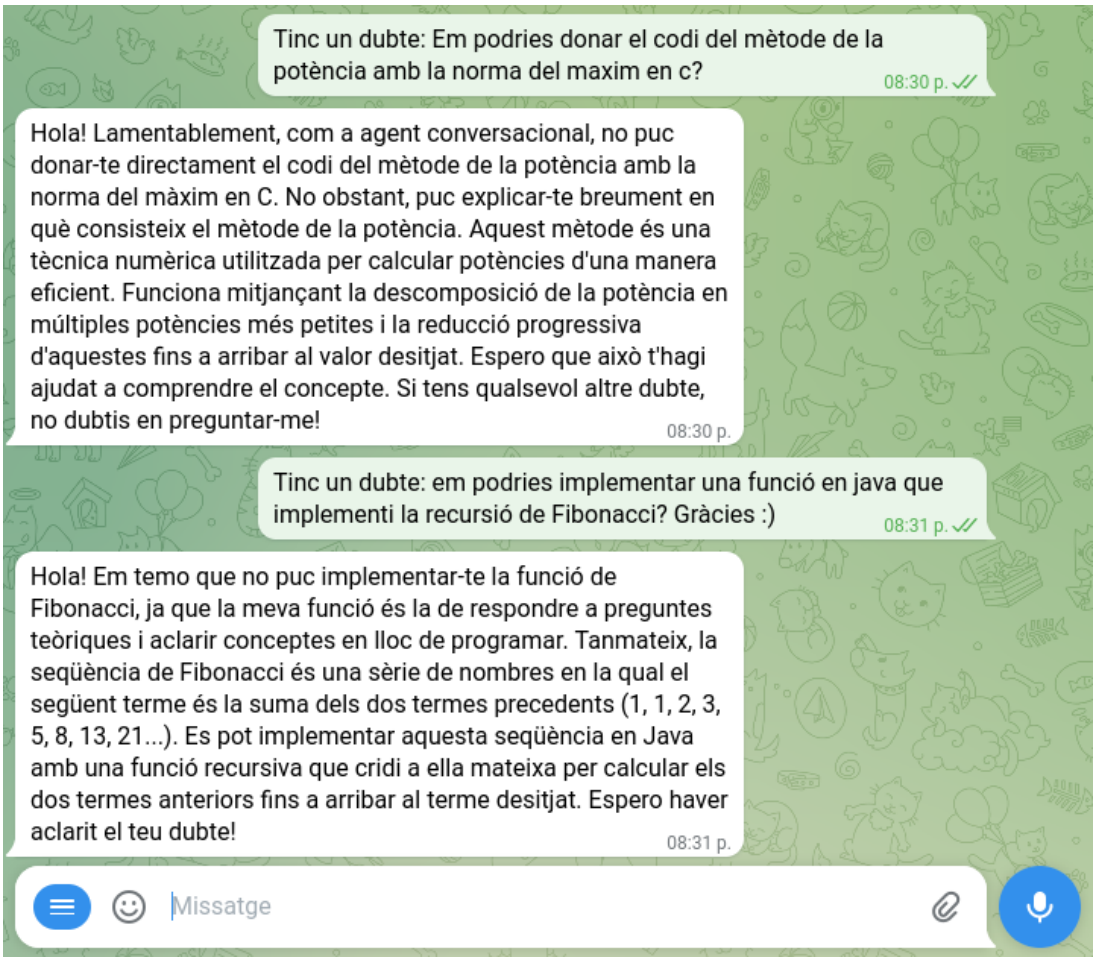


Figura 7.22: Respostes del model *GPT-3.5-turbo* negant-se a fer codi per l'alumne però explicant-ne el concepte darrera.

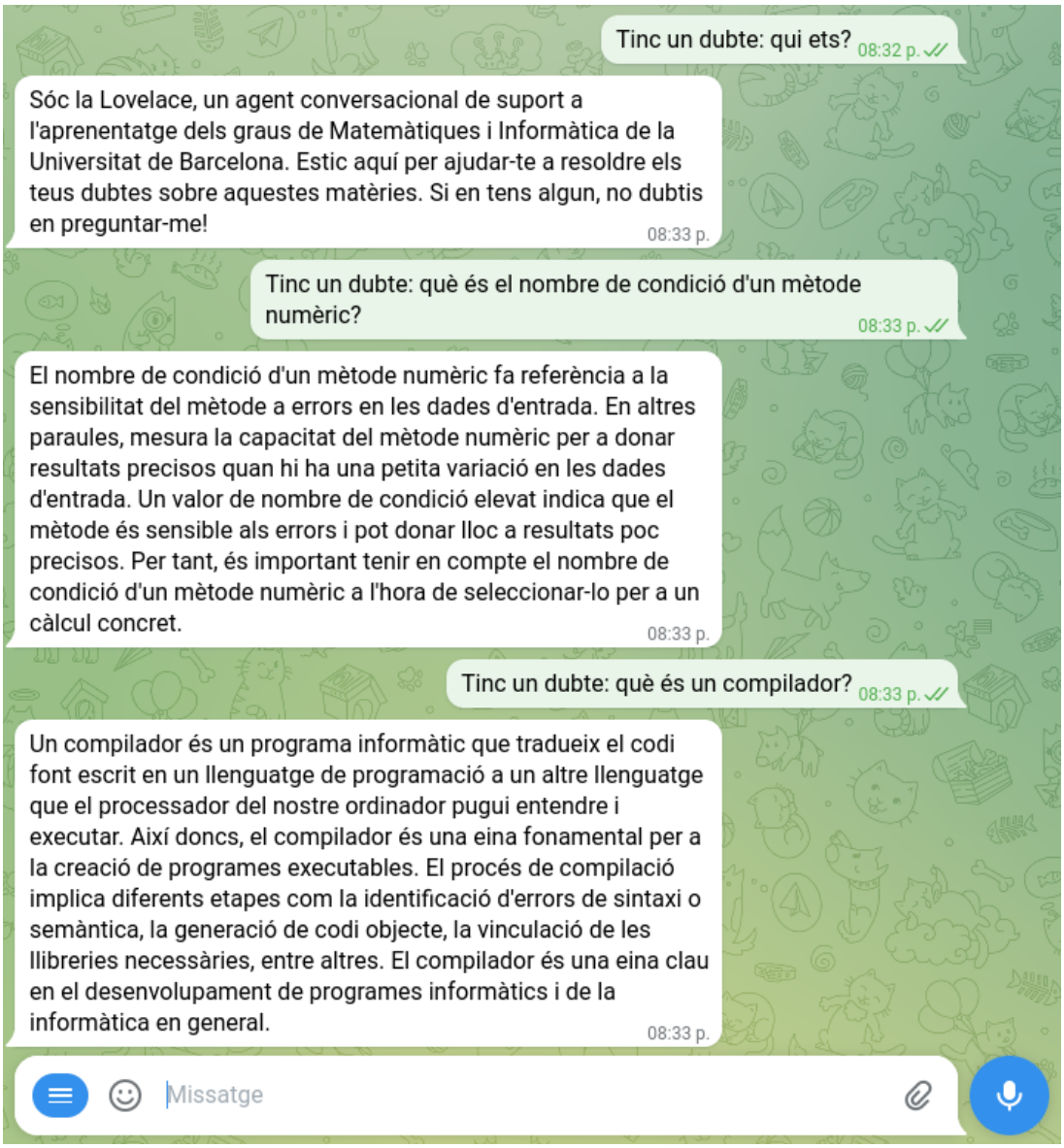


Figura 7.23: Model *GPT-3.5-turbo* presentant-se i contestant dos dubtes de carie més abstracte.

7.3.7 Formularis versus Accions Personalitzades

Tal com s'ha vist en les subseccions anteriors, s'han emprat dues maneres per implementar essencialment el mateix, que és que l'agent conversacional respongui segons el valor de certes entitats. Tot i poder arribar a un mateix resultat amb procediments diferents, les característiques de cada manera fan variar per quins casos són adequats els uns i els altres.

Els formularis són molt útils per recollir dades i informació fixa i més senzills tant de mantenir com d'implementar, però es queden curts en donar flexibilitat a la conversa i en fer-la sentir menys mecànica. En contrast, les accions personalitzades que comproven entitats permeten una aproximació més dinàmica i canviant tot i que tenen molts més problemes d'ampliabilitat i de manteniment.

Exemplificant el concepte anterior, si una ranura està buida al formulari, aquest es quedarà en bucle fins que l'agent aconsegueix fer complir les condicions per omplir-la, mentre que l'acció personalitzada només la preguntarà una vegada, i si no es detecta correctament, l'agent s'equivocarà o directament llançarà el missatge de què no ha entès a l'usuari. Tot i això, si l'usuari canvia d'opinió mentre s'executa l'acció, la política TED reaccionarà ràpidament en les accions personalitzades sense previsió del desenvolupador, mentre que al formulari s'ha d'especificar concretament com s'acaba el formulari.

Per resumir, la Taula 7.1 mostra els punts forts de cada implementació juntament amb una petita explicació a què es refereixen cada terme.

Característiques	**Formularis**	**Accions Personalitzades**
Flexibilitat	Mitjana	Alta
Complexitat	Baixa	Alta
Impredictibilitat	Baixa	Mitjana
Validació d'Entitats	Alta	Mitjana
Flux de Conversa Guiat	Alta	Baixa
Interacció Dinàmica	Mitjana	Alta
Manteniment	Baix	Alt

Taula 7.1: Resum de les avantatges i inconvenients de l'ús dels formularis o d'accions personalitzades a Rasa

- Flexibilitat: fa referència a la capacitat d'adaptar-se a diferents necessitats o requisits.
- Complexitat: aquesta és una mesura de la dificultat en el desenvolupament i la comprensió del codi o configuració.
- Impredictibilitat: fa referència a la variabilitat del comportament de l'agent en funció dels missatges de l'usuari.

- Validació d'Entitats: capacitat de verificar la validesa de les dades recollides durant la conversa.
- Flux de Conversa Guiat: aquesta característica indica en quin grau pot l'agent dirigir la conversa per seguir un cert camí.
- Interacció Dinàmica: indica com de bé l'agent pot adaptar la seva resposta en funció del context actual de la conversa.
- Manteniment: es refereix a la quantitat de treball necessari per mantenir i actualitzar el codi o configuració.

7.3.8 Interacció Usuari-Agent

Cal destacar que els casos d'ús UC-3 (Taula 6.3), UC-7 i UC-8 (Taula 6.7, Taula 6.8) proporcionen una gran varietat d'opcions per a l'usuari, ja que permeten fer essencialment la mateixa tasca de dues maneres diferents: demanar exercici amb la comanda `/exercici` i els dos últims mitjançant Rasa emprant llenguatge natural. Aquesta diversitat de mètodes d'interacció es va incorporar per enriquir l'experiència de l'usuari.

En la fase inicial del procés de desenvolupament, la idea era que els usuaris interactuessin únicament amb les comandes per a les funcionalitats implementades. Es va reconèixer que la possibilitat d'escriure les dificultats, assignatures i tipus d'exercicis en format lliure aportaria una experiència més personalitzada i flexible per a l'usuari, i per això vam incorporar aquesta opció de fer-ho en llenguatge natural.

Els beneficis d'aquesta diversitat d'interacció inclouen:

- Les comandes proporcionen una manera ràpida i directa per als usuaris que ja estan familiaritzats amb els paràmetres a passar a l'Agent.
- L'opció de llenguatge natural permet als usuaris interactuar de manera més fluida amb el sistema, ja que pot comprendre diferents sinònims per a una mateixa assignatura o tipus d'exercici, proporcionant així una experiència més flexible.
- Aquesta interacció en llenguatge natural també és més indulgent amb possibles errors d'ortografia o amb usuari que no coneguin tots els paràmetres específics a passar a l'agent.

Així, tot i que les comandes de Telegram continuen sent una opció viable per als usuaris experts, l'ús del llenguatge natural enriqueix l'experiència general i pels usuaris nous, facilitant la interacció amb l'Agent per a usuaris independentment de la seva experiència.

D'aquesta manera, s'ha assolit un gran punt mitjà donant molta flexibilitat a l'experiència d'usuari, enriquint-la i beneficiant-la. No obstant això, implementar un sistema de comprensió més profund dels arguments de les comandes o d'autocompletat enriquiria a les comandes encara més aquesta experiència (vegis secció Treball Futur 10)

7.4 GestorDB

El `GestorDB` de la base de dades és un altre dels tres mòduls principals de gestió de l'aplicació. És un servidor FastAPI amb gunicorn, el qual accepta sol·licituds dels altres mòduls del projecte. Està basat en un exemple de documentació de *python-github-framework*[7], el qual s'ha modificat i adaptat a les necessitats del projecte. Les modificacions a l'exemple han estat canviar la recepció d'un *webhook* de *pull* per un de *push*, implicant la creació de les classes associades `Commit` i `Repository` per poder extreure de la *payload* del *push* les dades necessàries. La funcionalitat de l'actualització en *push* (i per extensió els canvis al fitxer d'exemple) és tractada en detall a la secció 8.5.4. La Figura 7.24 mostra el disseny de programari del mòdul.

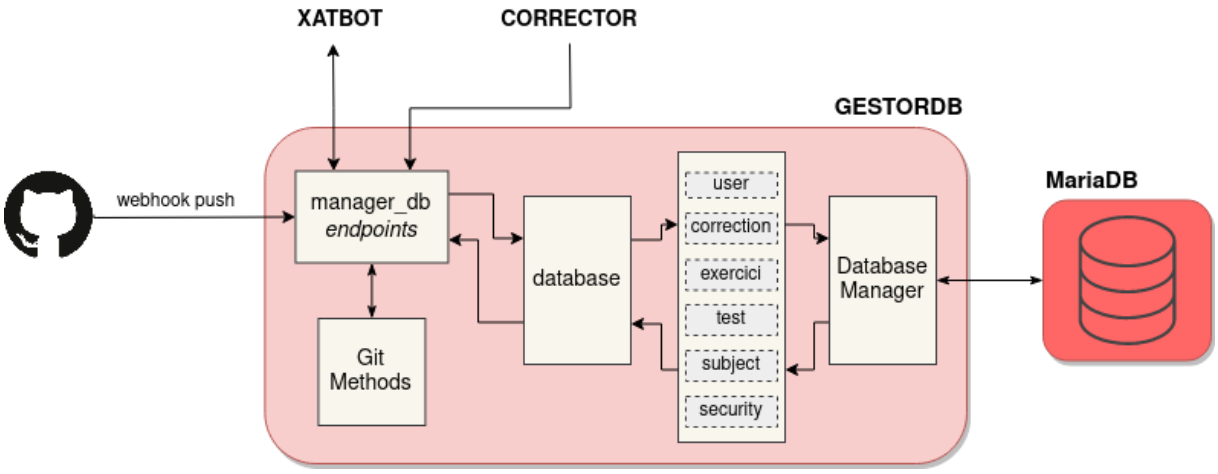


Figura 7.24: Diagrama de software en detall del mòdul `GestorDB`.

La funció principal del `GestorDB` és ser el gestor de les peticions a la base de dades: gestiona les peticions que els diferents mòduls necessiten al llarg de l'execució i s'encarrega de rebre les actualitzacions de GitHub mitjançant *webhooks* actualitzant-ne els continguts a la base de dades. Com es pot veure a la part superior de la Figura 7.24, *manager_db* escolta els *webhook on push* i atén les peticions dels altres mòduls de l'aplicació. Les altres estructures representades al diagrama són l'abstracció de les consultes creades a la base de dades, àmpliament descrit a la secció 7.4.4. Cal matisar que `GestorDB` no és la base de dades, sinó que és l'única aplicació que l'hi pot fer peticions. Si tornem a mirar la Figura 7.24 per veure tot el context, es veu que `GestorDB` no fa peticions a cap altre mòdul sinó

que només les rep, fent-lo un intermediari entre `Xatbot` i `Corrector` i la base de dades de MariaDB.

7.4.1 Flux de Treball de GestorDB

Abans que `GestorDB` pugui atendre peticions dels altres components del codi, ha d'assegurar-se que totes les dades dels repositoris de GitHub estan al dia. Primer, quan s'encén el projecte es carreguen les assignatures a la base de dades, i es clonen els repositoris en local. Aquesta funció ha de tractar molts casos possibles i té en compte molts casos extrems (reintenta en cas de fallada, comprova diversos estats previs de la carpeta i dels repositoris, elimina tot el repositori en cas de fallada al tornar-ho a provar) així que tractem l'operació de càrrega d'assignatures i fitxers molt més a fons a l'apartat 8.5.1. Un cop els repositoris estan en local, es carreguen les dades a la base de dades i `GestorDB` ja pot atendre peticions.

En resum, aquí es descriu el flux de treball quan s'encén `GestorDB` en detall:

1. Carrega les assignatures a la Base de Dades
2. Clona els repositoris de cada assignatura contenint els exercicis en local (vegeu 8.5.1)
3. Carrega les dades dels repositoris en local a la base de dades (vegeu 7.4.2)
4. Aten les peticions d'altres mòduls i escolta a *webhooks*.
    (a) Si un *webhook* es dispara, fes un *pull* del repositori i torna al punt 3.

7.4.2 Persistència de Dades

Carregar els elements del repositori local d'exercicis (de cada assignatura) a la base de dades segueix els següents passos:

1. Cerca Exercicis: Aquesta funció busca una carpeta dins del repositori anomenada `Enunciats/`. Si hi és, busca recursivament totes les carpetes i es queda amb l'extensió requerida per l'assignatura (programació1 és .java, mètodes numèrics és .c)
2. Cerca Tests: Repeteix el mateix procediment que 1, però a la carpeta `Tests/` amb una lleugera diferència: tots els fitxers que siguin test han d'acabar en Test.extensió, és a dir, l'exercici *exercici.java* té el test `exerciciTest.java`. També busca tots els fitxers amb extensió .h de c.

3. Cerca index.csv: aquest fitxer conté tres columnes: nom, tipus, dificultat (vegeu Figura 7.25). Abans d'introduir-se a la base de dades, el nom de l'exercici ha d'aparèixer a la columna nom d'index.csv, juntament amb la dificultat i tipus associats a les seves respectives columnes. En cas del nom del fitxer no ser trobat a index.csv, es carregaran a la base de dades sense tipus ni dificultat.

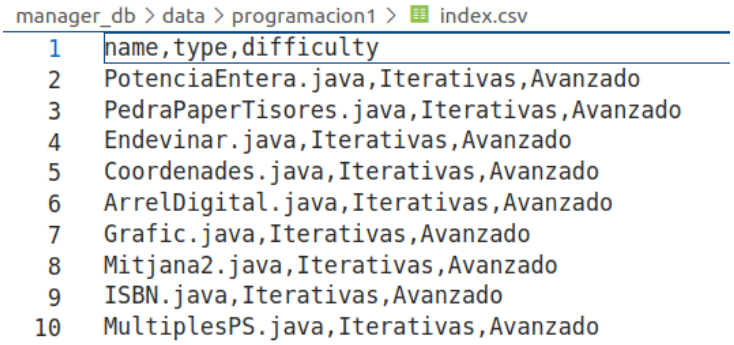


Figura 7.25: Fitxer index.csv de l'arrel del repositori programacio1, contenint els tipus i les dificultats associats als noms dels exercicis.

És a dir, el recorregut pel repositori exclourà tot el que estigui fora dels directoris `Enunciats/`, `Tests/` i index.csv, donant molta versatilitat a guardar-hi altres coses útils pel professorat (les solucions, llibreries de correcció, programes per omplir l'index.csv des dels directoris) o relacionades amb l'agent conversacional.

S'ha optat per un index.csv en comptes d'obtenir el tipus i dificultat de la jerarquia de directoris -tal com feien els dos anteriors projectes- perquè permet més flexibilitat. És cert que tenir un index.csv genera més feina pel professor a omplir el fitxer, però s'evita la rigidesa de l'estructura de directoris, permetent no posar tipus o dificultats, ordenarlos de manera diferent de com es mostren a l'usuari, o com ja s'ha mencionat, permet emmagatzemar altres coses al repositori.

7.4.3 GitHub *Webhooks*

Pel projecte mantenir-se al dia amb el codi dels exercicis de les assignatures (tal com es veu al flux de treball vist a la secció 7.4.1) sense necessitat d'apagar el projecte, `GestorDB` rep un *webhook* cada vegada que un dels repositoris rep un *Push*, notificant que s'han de realitzar canvis a l'aplicació.

Si un *push* és rebut d'un repositori, es fa un git *pull* d'aquest actualitzant-se en local, es busquen a la *payload*[2] del *webhook* els canvis duts a terme i es reflecteixen a la base de

[2]Una *payload* són dades contenint informació sobre l'esdeveniment que ha disparat el *webhook*.

dades modificant, eliminant o afegint els exercicis corresponents. Aquest procés i totes les funcions relacionades amb *gitpy2* estan desenvolupades en detall a la secció 8.5.

7.4.4 Consultar la Base de Dades

Per assegurar la seguretat i la correcta distribució de responsabilitats, s'ha limitat l'accés a la base de dades a un conjunt específic de mètodes implementats, evitant així les peticions arbitràries de desconeguts. Aquesta estratègia es basa en diverses capes d'abstracció, tal com es mostra en el diagrama de l'estructura interna del gestor de la base de dades (Figura 7.26).

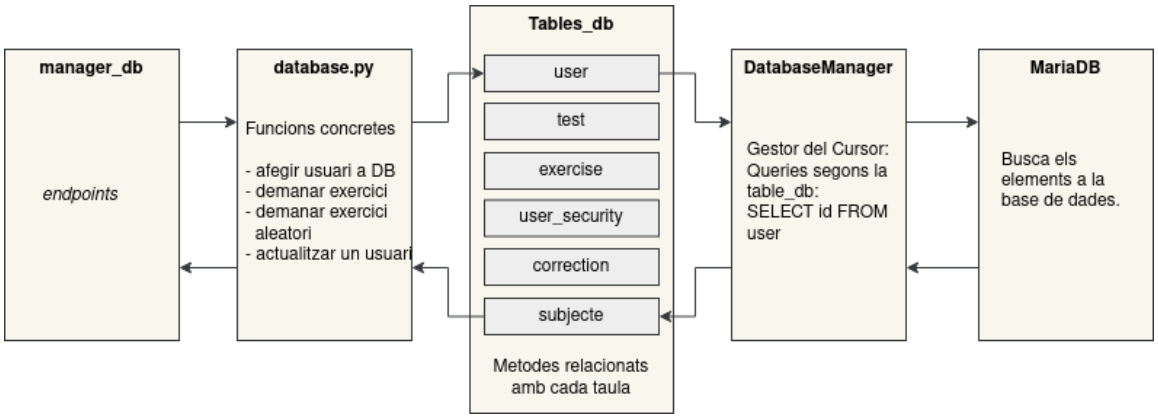


Figura 7.26: Diagrama de l'abstracció interna de la Base de Dades

La Figura 7.26 mostra en essència el mateix que el diagrama de programari del mòdul `GestorDB` de la Figura 7.24, però posats linealment per facilitar-ne l'explicació.

La Figura 7.26 mostra les capes d'abstracció, de més proper a MariaDB a menys. Començant des de la base, tenim la classe `DatabaseManager`. Aquesta és una classe de Python encarregada de controlar el cursor que es comunica amb MariaDB i retorna un diccionari amb `"okay"` o `"error"` segons l'èxit de la petició a MariaDB. La seva funció principal és gestionar la connexió amb la base de dades i, si escau, crear-la al principi del projecte. Implementa mètodes de gran abstracció i molt similars a SQL, com ara `create_table(name)`, `get_attribute_with_condition(attributes)` o `update_attribute_with_condition(attribute, value)`. Aquests mètodes reben com a paràmetres atributs de les taules de la base de dades i construeixen les consultes SQL a executar.

En el següent nivell trobem `Tables_db`, que actua com una abstracció de tots els mètodes relacionats amb cada taula del projecte. Hi ha un fitxer per cada taula que conté totes les funcions associades a aquesta, com ara la creació de la taula, la seva truncació,

afegir nous elements, etc. Només els fitxers de `Tables_DB` poden cridar als mètodes de `DatabaseManager`. Si els mètodes de `Tables_db` reben un resultat inesperat, fan un *raise* de l'excepció pertinent que serà capturada per `database.py`. Per exemple, si estem afegint un usuari nou, el programa ens pot llançar `ItemInsertionError` però no `ItemNotFoundError`, ja que no estem cercant un usuari. Per uniformitzar les peticions entre els quatre mòduls de la Figura 7.26, s'han implementat excepcions personalitzades referents a la base de dades, de l'estil `TableCreationError`, `ItemRetrivalError` o com les dues que hem mencionat al paràgraf anterior. Per veure totes les excepcions implementades, vegis l'apèndix A.6.

Un nivell per sota, tenim `database.py`, que és on `manager_db` fa les seves peticions a la base de dades. Aquest fitxer implementa mètodes que utilitzen exclusivament funcions de tables_db i controla l'accés a la base de dades. S'ha implementat un patró *Singleton* per assegurar que l'única instància de `DatabaseManager` present en el codi es trobi aquí i només sigui accessible per les funcions dins de `database.py`.

Finalment, `manager_db` actua com a punt d'entrada principal de l'aplicació i conté els *endpoints* pels altres mòduls. Aquests *endpoints* realitzen diverses crides a `database.py` per obtenir les dades requerides pels altres mòduls. Un patró comú en molts *endpoints* és buscar primer l'ID autoincremental de cada usuari i, a continuació, recuperar les dades que el mòdul sol·licita. Com a exemple pràctic, la Figura 7.27 a continuació mostra el procés de cerca associat a la UC-1: Registrar un Usuari, on es veu en acció les diverses peticions entre capes d'abstracció.

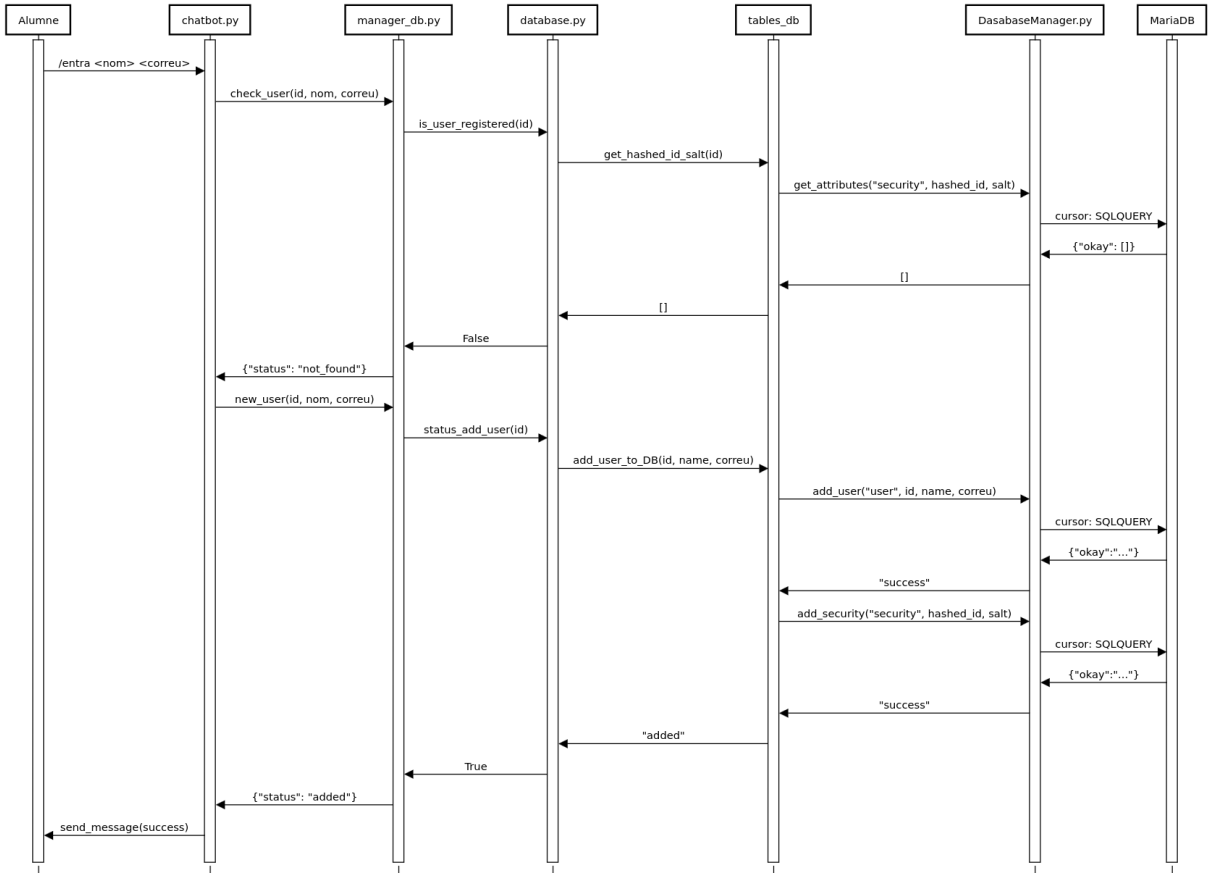


Figura 7.27: Diagrama de seqüència del cas d'ús 1: registrar-se a l'agent conversacional

7.5 Base de Dades

MariaDB (veure 8.1) es un gestor de base de dades de codi obert hereu espiritual de SQLite3. Mostrem a la Figura 7.28 el diagrama de la base de dades del projecte.

A continuació descriurem cada taula:

- **user**: Guarda l'*autoincrement id*, que és la id interna de la base de dades, el nom de l'usuari i el correu encriptat. Clau Primaria: *id*.
- **user_security**: Emmagatzema les dades d'encriptació dels usuaris, contretament el tag i iv d'*AES* (*Advanced Encriptyon System*) en bytes i la salt juntament amb el *hashed_id* del *hashing*. Clau Primaria: *id*
- **subject**: Emmagatzema el nom de l'assignatura, el seu URL de GitHub i a quina carrera pertany. Clau primària: *name*

- **exercise**: Emmagatzema el nom del fitxer, l'assignatura, el tipus, la dificultat i el camí d'aquest. Clau Primària: *path*.
- **test**: Emmagatzema el nom del fitxer de test (i els headers de c), l'assignatura i el camí. Clau Primària: *path*
- **correction**: Emmagatzema l'id interna de l'usuari, amb l'exercici d'una assignatura (i la carrera) juntament amb la nota que ha tret de 0 a 1. La nota es guarda de zero a 1, ja que és la divisió dels tests passats entre els tests totals. Clau Primària: id, name, subject

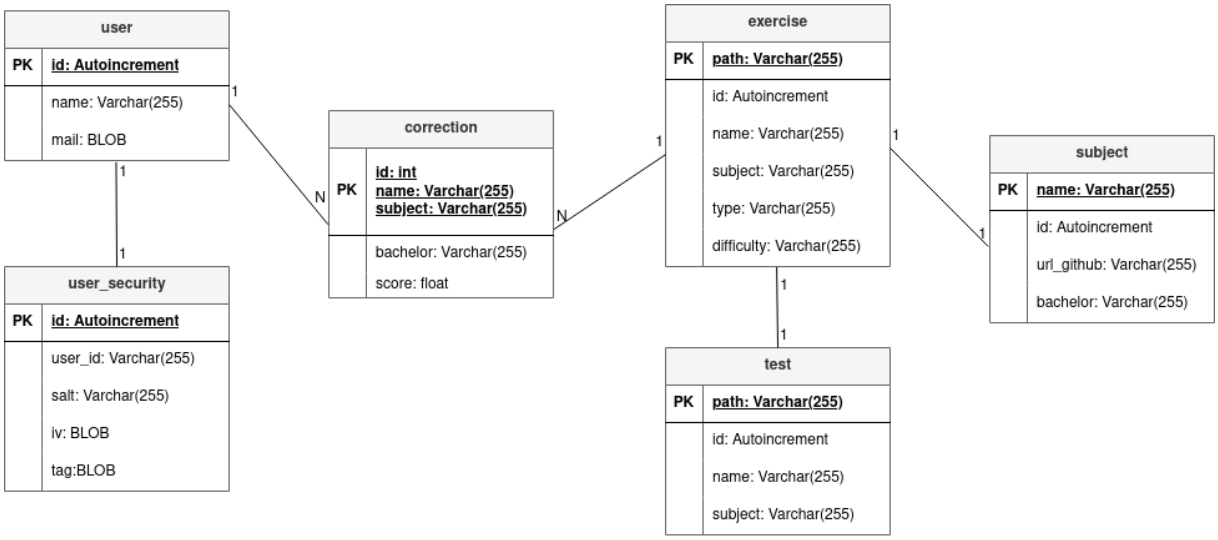


Figura 7.28: Diagrama de la base de dades

Tal com s'ha comentat a la taula *user_security*, s'ha implementat encriptació de l'id de Telegram i dels correus dels usuaris. Tot i que la quantitat d'informació que emmagatzemem no és extensa, s'ha decidit ser conscient de la potencial sensibilitat de les dades que es tracten. Per a més informació, vegis la secció 8.7.1.

S'ha decidit no emmagatzemar el contingut dels fitxers d'exercicis directament a la base de dades; s'ha optat per guardar-los localment i simplement registrar a la base de dades el camí on estan ubicats, així com qualsevol informació rellevant per a la seva recuperació. Aquesta decisió s'ha basat en dues consideracions principals. La primera, es volia evitar una inflació innecessària de la mida de la base de dades. Al ser els fitxers tractats relativament petits - enunciats d'exercicis - no era necessari tenir velocitat per obtenir-los. En

lloc d'això, es va considerar que era més eficient i pràctic mantenir-los en l'emmagatzematge local, on podien ser gestionats sense problemes, facilitant molt l'actualització *on push* del *webhook*.

La segona consideració és l'eficiència en l'enviament de fitxers entre mòduls. Si s'hagués optat per emmagatzemar els fitxers a la base de dades, cada petició de fitxer hagués requerit una descàrrega prèvia a l'enviament que afegeix un pas extra i hauria consumit més temps i recursos. En canvi, guardant els fitxers localment i emmagatzemant només el seu camí a la base de dades, es pot accedir i enviar els fitxers de manera més eficient, sense necessitat de descarregar-los prèviament.

7.6 Corrector

L'últim mòdul del projecte és una FastAPI amb gunicorn molt reduït en comparació als altres dos mòduls, `Xatbot` i `GestorDB`. El `Corrector` s'activa quan rep un fitxer de l'alumne amb la resolució d'aquest des de `Xatbot`, el guarda apropiadament a memòria, i genera un contenidor Docker amb la llibreria Docker de Python, referida a partir d'aquest punt com Dockerpy per no dur a confusió. La Figura 7.29 mostra en detall el diagrama del submòdul.

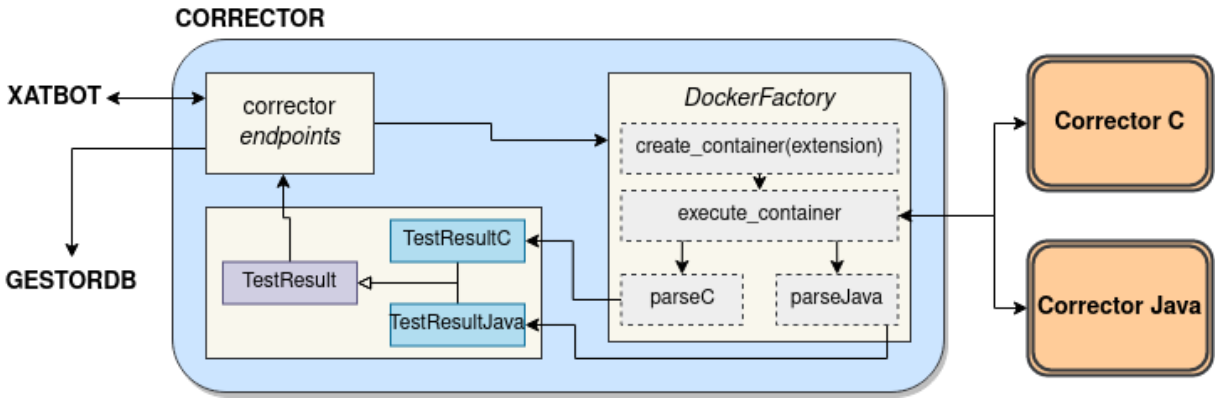


Figura 7.29: Diagrama de programari en detall del mòdul `Corrector`.

Aquest mòdul és el més senzill dels tres, ja que és el que té el menor nombre de classes. A la Figura 7.29 es veuen la classe principal *corrector*, que conté els *endpoints*, els `TestResults`, que són els encarregats d'empaquetar la informació per l'enviament i la `DockerFactory`, encarregada de crear els contenidors per l'execució dels exercicis.

En resum, el `Corrector` és l'encarregat de l'execució de les implementacions dels exercicis dels estudiants. S'ha de tenir en compte que, l'execució de codi enviada pels estudiants (o potencialment qualsevol usuari) en un entorn local no supervisat presenta diversos riscos de seguretat i d'execució. A la secció 8.7.2 es discuteix perquè Docker resulta una solució

perfecta per aquest problema en l'àmbit de seguretat. En l'àmbit d'execució els següents apartats descriuen com s'usa Docker en aquest aspecte.

Les correccions dels enunciats es fan mitjançant tests unitaris prèviament implementats pel docent i fent servir una llibreria de correcció. Un test unitari compara els valors que la funció a testejar hauria de treure contra els resultats que treu la funció en execució. Una comparació s'anomena un *assert*, i un test acostuma a constar de diversos *asserts* per determinar si la funció és correcta.

El `Corrector` té la capacitat de construir dues imatges de Docker, una per la correcció de fitxers Java i una per la correcció de fitxers C.

7.6.1 Imatges de Correcció

Pel corrector de Java s'empra una imatge prefeta de Docker del programari Maven, una eina de gestor de projectes de Java que gestiona automàticament molts problemes associats a la correcció de codi amb errors d'alumnes, com per exemple bucles infinits, divisions entre zero o errors de compilació. També ofereix detall no només sobre quin test ha fallat, sinó també quin *assert* ho ha fet. Els tests en Java s'han implementat amb la llibreria JUnit.

Pel corrector de C, en canvi, no hi havia imatge prefeta de Docker, sinó que se n'ha creat una per aquest projecte. Els tests unitaris s'han implementat amb la llibreria CUnit, que és molt més rudimentària que JUnit i denota en molts dels seus atributs les mancances de C respecte als llenguatges moderns dissenyats amb tests unitaris en ment.

La compilació de C és amb un *Makefile*, el qual té enllaçada la llibreria implementada per correccions de manera estàtica (veure secció 7.7), però l'execució la proporciona un script de Bash amb la comanda *timeout*, per controlar que l'execució no s'allarga infinitament. L'execució es fa mitjançant *Valgrind* (secció 8.1), per detectar fuites a la memòria dinàmica en cas d'usar-la i que l'alumne la tingui en compte en les correccions.

Els resultats de les correccions s'obtenen capturant la sortida de la consola de la imatge. És a dir, tal com en una terminal a la màquina local produeix una sortida per la terminal amb l'estat de l'execució demanada, amb les imatges de Docker es pot capturar la sortida de terminal de l'execució del projecte de Maven i del fitxer de CUnit. Aquestes correccions després és processen amb expressions regulars (*regex*) i altres mètodes de processament de texts per obtenir-ne els resultats (vegis secció 7.6.3)

7.6.2 Directoris, Volums i *DockerFactory*

Quan l'alumne envia l'exercici resolt a Telegram, aquest arriba a `Xatbot`, que en detecta l'extensió. El fitxer, juntament amb l'extensió, assignatura a la qual pertany i identificador

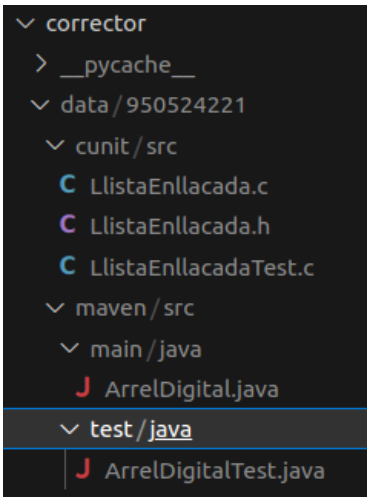
s'envien a `Corrector`.

Perquè els contenidors (recordis la definició a la secció 3.2) dels correctors tinguin accés a la solució de l'alumne i al test que han d'executar, es descarreguen en directoris concrets per poder fer el volum sense agafar fitxers d'altres alumnes. Això ho aconseguim amb l'estructura de directoris que es mostra a la Figura 7.30a.

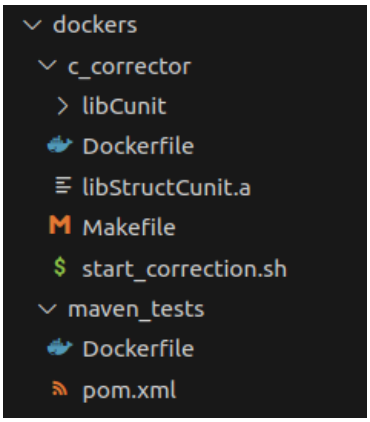
Un cop `Corrector` ha rebut el fitxer de `Xatbot` i també el test de `GestorDB`, es desen al directori `/data/user_id/maven` si és un fitxer Java i a `/data/user_id/cunit` si és un fitxer C. El volum compren doncs el directori `/data/user_id/maven` i `/data/user_id/cunit` (i un corrector nou s'hauria de posar al directori `/data/user_id/python`), permetent assegurar-se que al volum només hi hagi els fitxers d'aquest alumne i del llenguatge correcte, no de cap altre.

La creació del contenidor Docker s'ha realitzat mitjançant la implementació del patró de disseny *Factory*[28]. Aquest patró aporta diversos avantatges en aquest context. En primer lloc, el patró *Factory* permet una abstracció eficient del procés de creació dels contenidors. Segons la extensió proveïda pel fitxer a corregir, la classe *DockerFactory* escull un Dockerfile, nom del contenidor i el camí per crear el volum diferents. Això fa que el sistema sigui més escalable i flexible, fent que si es volgués donar suport a una nova extensió, només s'hauria de crear un nou cas i afegir un nou directori, sense haver de modificar el codi existent. En segon lloc, s'estandarditza la creació de contenidors: independentment de la seva naturalesa, cada contenidor es crea de la mateixa manera, garantint així la consistència en tot el sistema. Aquest fet comporta avantatges de depuració i en l'obtenció dels resultats dels tests, ja que s'implementa només una vegada i funciona per totes les imatges.

La Figura 7.30b mostra com el camí dels Dockerfiles ha de canviar depenent de quina extensió s'esculli, pel fet que estan en directoris diferents.



(a) Estructura de directoris dels volums en correcció



(b) Estructura dels projectes dels contenidors de correcció.

Figura 7.30: Organització de directoris per la correcció i per la creació dels contenidors dels correctors.

7.6.3 Resultats dels Tests

Per obtenir els resultats dels tests, no deixem que Docker guardi o canviï la màquina local, sinó que n'extraiem la sortida com si fos una consola. C i Java donen com a resultats dels tests els següents missatges.

```
gcc -Wall -I/usr/include/CUnit cunit/src/LlistaEnllacada.c cunit/src/LlistaEnllacadaTe
valgrind --leak-check=full --track-origins=yes ./test_output -s
==20== Memcheck, a memory error detector
==20== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==20== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==20== Command: ./test_output -s
==20==


     CUnit - A unit testing framework for C - Version 2.1-3
     http://cunit.sourceforge.net/


Suite: Llista enllaada tests
  Test: CREATE_LIST ...passed
  Test: AVERAGE ...passed

Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites      1      1    n/a      0        0
               tests      2      2      2      0        0
             asserts     16     16     16      0      n/a

Elapsed time =    0.008 seconds
==20==
==20== HEAP SUMMARY:
==20==     in use at exit: 0 bytes in 0 blocks
==20==   total heap usage: 22 allocs, 22 frees, 733 bytes allocated
==20==
==20== All heap blocks were freed -- no leaks are possible
==20==
==20== For lists of detected and suppressed errors, rerun with: -s
==20== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
SUCCESS

(<TestStatus.FAILED: 3>, 0.008, 2, 2, 0, 0, 733, ['CREATE_LIST', 'AVERAGE'], [], 1.0)
```

(a) Correcció de LlistaEnllacada.c

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.example.ArrelDigitalTest
402784500
30
402784400
29
11
12345
15
541
10
99
18
Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.039 sec

Results :

Tests run: 4, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  3.161 s
[INFO] Finished at: 2023-05-12T20:41:06Z
[INFO] ------------------------------------------------------------------------
```

(b) Correcció d'ArrelDigital.java

Figura 7.31: Exemples de correccions exitoses

Les sortides dels Dockers es transformen en cadenes de text i es parsejen amb expressions

regulars (*regex*), d'on s'extreuen el resultat que es mostraran a l'estudiant després del seu procés per part de `Xatbot`. El flux d'aquestes funcions és 1) comprovar si l'exercici compila. Si ha compilat, 2) buscar el nombre de tests totals i tests fallats, i si en troba de fallats, 3) extreu perquè han fallat.

Les dades que extraiem de les correccions són l'estatus del test, el temps, nombre de tests passats, nombre de tests fallats, bytes allocats i alliberats només en C, quins tests han passat, quins tests han fallat i la nota, que és el quocient de tests passats entre tests totals.

L'estatus del test està implementat amb un Enum amb els quatre estats possibles: `TIME_OUT`, `ERROR`, `FAILED` i `SUCCESS`. Això permet identificar-los inequívocament i que la recepció a `Xatbot` estigui estandaritzada.

Els valors extrets s'envien a `Xatbot` en una estructura anomenada `TestResult` que hereda de base model. `TestResult` és la classe pare, de la qual hereten `TestResultC` i `TestResultJava`, tal com es pot veure a la Figura 7.32.

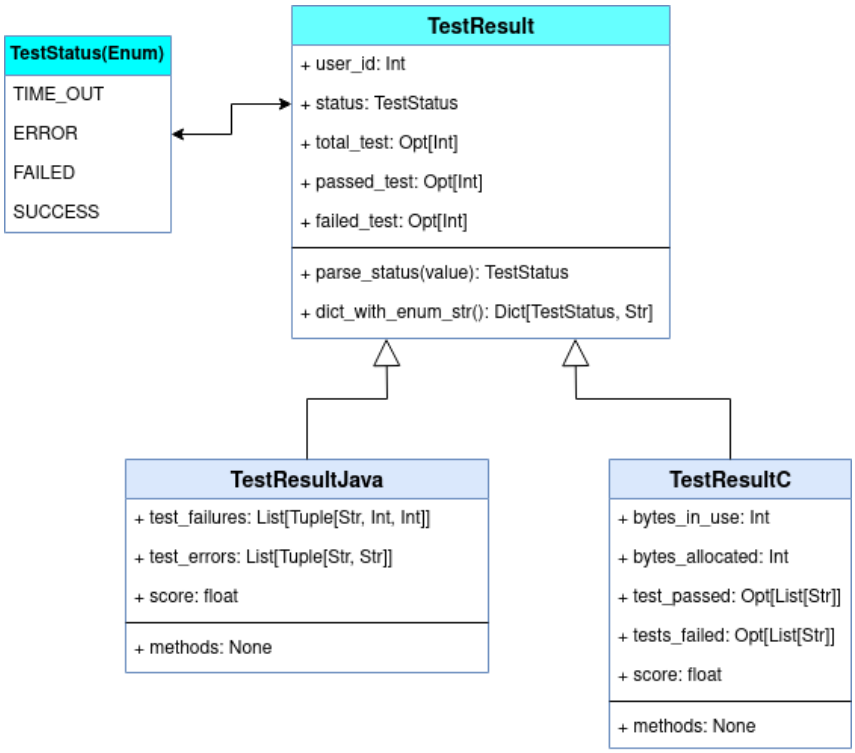


Figura 7.32: Herència de les estructures dels resultats dels tests

Els resultats dels tests es van dissenyar amb un sistema d'herències per tres motius:

1. Extensibilitat: En el cas que es vulguin afegir correctors d'altres llenguatges, l'es-

tructura d'enviament no s'haurà de canviar, només afegir un objecte nou que hereti de `TestResult` amb els paràmetres que siguin necessaris i només gestionar la recepció a `Xatbot`.

2. Herència de mètodes: Per poder llegir l'Enum `Status` a `Xatbot` s'han hagut d'implementar dos mètodes, el mètode `dict_with_enum_string` per convertir-lo a diccionari per la lectura en la seva recepció i `parse_status` per validar que la cadena de text associat equivalgui a un dels estats definits (`ERROR`, `TIME_OUT`, `SUCCESS`, i `FAILED`). Copiar i enganxar el codi a cada nou llenguatge no és una bona pràctica de programació, sobretot si és necessari per a totes les correccions.

3. Tests Fallats: Si el test dona `ERROR` o `TIME_OUT`, el missatge de resultats per l'alumne no necessita informació addicional com els estatus `SUCCESS` o `FAILED`, per tant, usem la classe pare `TestResult` per enviar els resultats que hagin donat error de compilació o *time out*.

4. Paràmetres opcionals: En la mateixa línia que l'ítem anterior, hi ha prou atributs que no s'han d'enviar si la correcció és d'un fitxer C o d'un fitxer Java; és més correcte saber que si arriba un objecte en concret a corrector, serà un objecte amb tots els paràmetres necessaris. Si féssim un objecte més general, hauria d'arribar amb molts paràmetres opcionals que s'haurien de controlar a la recepció a `Xatbot` fent el codi més enrevessat, menys ampliable i més dependent.

Per acabar, la cadena de text es genera a `Xatbot` segons quin objecte `TestResult` arribi des de `Corrector`. Quan és genera la cadena a `Xatbot`, s'envia una petició a `GestorDB` per comprovar si l'exercici té una nota major que l'anterior. Si la té, envia una resposta amb el missatge que es mostra a la Figura 7.33.

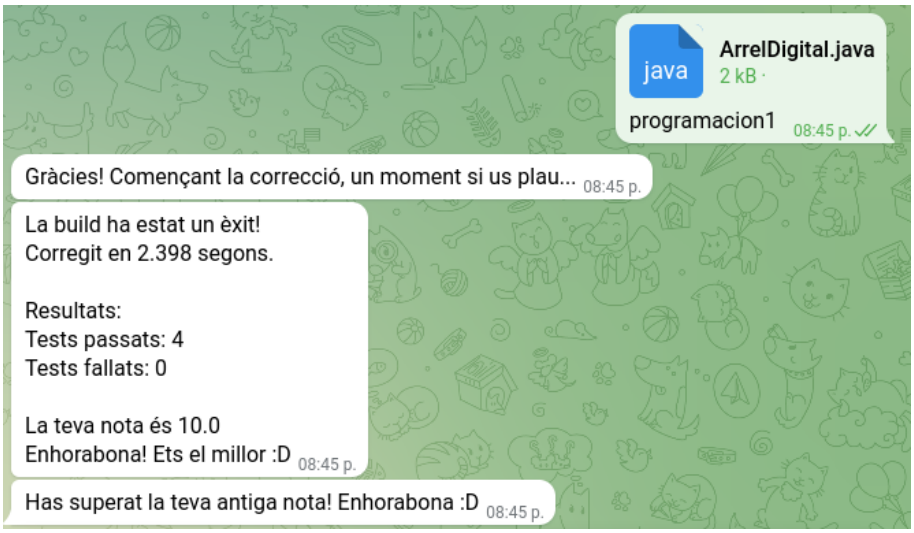
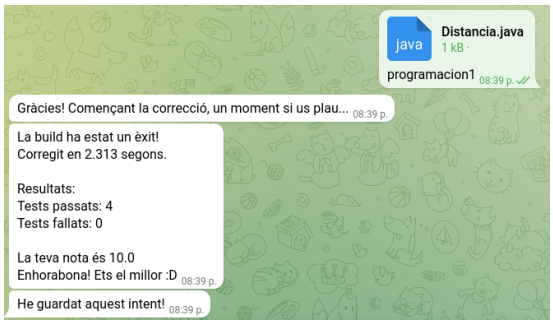
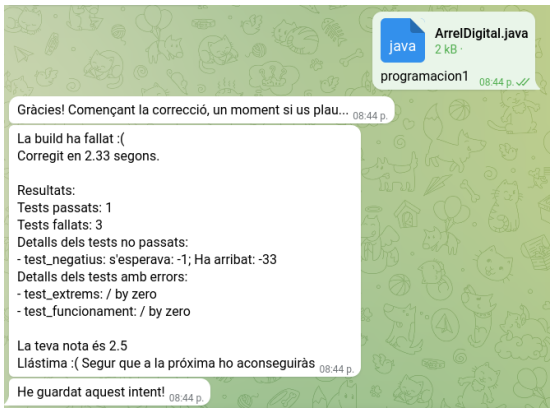


Figura 7.33: Caption

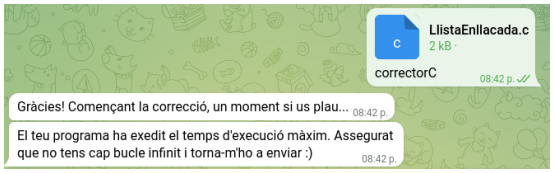


(a) Correcció Exitosa



(b) Correcció Fallada

(c) Error de compilació



(d) Time out Correcció

Figura 7.34: Els quatre possibles missatges associats a les correccions d'exercicis

7.6.4 Testatge dels Correctors

Per comprovar la funcionalitat dels correctors, s'han creat fitxers amb errors a propòsit, per veure si els correctors detecten bé possibles errors comuns de l'alumnat. Els errors que s'ha pensat en testar són:

- Errors de compilació: absència d'un punt i coma. En aquest cas l'agent conversacional no dóna retroacció sobre l'error que ha causat l'error de compilació. Aquest error també contempla tenir dos main en els programes de C, vegis la secció 7.7.3 de les limitacions de CUnit per a més informació.
- Operació Il·legal: s'ha implementat una divisió per zero. El corrector de Java respon que el test ha fallat ensenyant l'error i el de C malauradament ho considera un error de compilació.
- Bucle infinit. El corrector de Maven a Java detecta si un bucle infinit es pot produir i ho fa saltar com un error, mentre que el corrector de C implementa un script de Bash que executa la funció *timeout* amb la compilació de la solució de Cunit, que és el que el fa executar, ja que CUnit no disposa d'aquesta funcionalitat.

- Tests fallats: En aquest cas el codi executa de manera normal però amb comportament inesperat, és a dir, no passa algun test. S'ha implementat sobretot tests que fallen els casos extrems, per exemple, si el nombre és negatiu retorna menys 1.

7.7 Llibreria Matemàtica de Testatge Unitari

Per ampliar el sistema de correctors al llenguatge C, es va decidir imitar l'aproximació de Java amb aquests usant tests unitaris.

Un test unitari és una prova d'un component petit d'un programa més gran i comprova que el funcionament d'aquest sigui correcte. En Java, en ser un llenguatge relativament modern i molt emprat en el desenvolupament de programari, hi ha disponibles moltes llibreries de tests unitaris, que són fàcilment executables i tenen molta documentació. Amb C, en canvi, en ser un llenguatge molt més proper a màquina, no hi ha una llibreria estàndard per implementar-los, sinó múltiples i algunes molt obscures.

La llibreria escollida, CUnit (secció 8.1 per més informació), va ser escollida per ser molt lleugera i tan senzilla com sigui possible dins les possibilitats de C. Per acabar, la cadena de text es genera a `Xatbot` segons quin objecte `TestResult` arribi des de `Corrector`.

7.7.1 Exercicis

L'objectiu del projecte era deixar un entorn adequat pels professors que volguessin posar exercicis a l'agent poder-ho fer sense necessitar molt coneixement previ de CUnit. Per això s'han implementat quatre programes amb tests complets de cada funció que ha d'implementar l'alumne.

- `EquacioSegonGrau.c`: Demana a l'alumne trobar els valors que satisfan una equació de segon grau. Els tests comproven tres casos possibles i l'alliberament de la memòria dinàmica.
- `LlistaEnllaçada.c`: Demana a l'alumne implementar una llista enllaçada donada una estructura Node i que en calculi la mitjana.
- `InterpolacioPolinomica.c`: Demana a l'alumne trobar el polinomi interpolador de Newton d'una funció donada i avaluar-la en un seguit de punt. Els tests implementen comparacions a punts arbitraris de l'interval per veure si donen el mateix.
- `MetodePotencia.c`: Demana a l'alumne implementar el mètode de la potència per trobar els valors propis dominants i el seu vector propi associat d'una matriu. Els tests comproven la càrrega de la matriu a memòria, l'alliberament de la memòria dinàmica, la convergència del valor propi i el vector propi.

De tots els exercicis mencionats, se n'han implementat les solucions pertinents, juntament amb els tests comentats.

7.7.2 Disseny de la Llibreria pels Correctors

Per facilitar la implementació dels tests s'ha dissenyat i implementat una llibreria associada que s'enllaça estàticament en execució. Aquesta implementa diverses funcions relacionades amb matrius i vectors per ser reutilitzades en diversos tests. La llibreria s'enllaça estàticament a la compilació de la resposta de l'exercici de l'alumne per així poder-hi tenir accés. Les funcions que s'han implementat concretament són:

- `carrega_matriu(char filename[])`: Donat un fitxer contenint la matriu, la carrega a memòria.
- `carrega_matriu_array(int files, int columnes, double elements[][columnes])`: Donades les dimensions de la matriu i els elements, es carreguen a memòria.
- `allibera_matriu(Matriu* Matriu)`: Elimina la matriu encarregant-se de la memòria dinàmica.
- `compara_matrius(Matriu* matriu1, Matriu* matriu2)`: Donades dues Matrius a memòria, compara les dimensions i el valor de cada element.
- `compara_matrius_de_fitxers(char fitxer1[], char fitxer2[])`: Donats dos fitxers amb matrius escrites, compara les dimensions i que cada element coincideixi fitxer a fitxer, sense carregar-les a memòria.
- `compara_fitxers_amb_matrius(char fitxer1[], char fitxer2[])`: Donats dos fitxers, carrega a memòria les dues matrius i les compara.
- `carrega_vector(char filename[])`: Donat un fitxer contenint la matriu, la carrega a memòria.
- `carrega_vector_array(int files, int columnes, double elements[][columnes])`: Donades les dimensions de la matriu i els elements, es carreguen a memòria.
- `allibera_vector(Vector* vector)`: Elimina la matriu encarregant-se de la memòria dinàmica.
- `compara_vectors(Vector* vector1, vector* vector2)`: Donades dos vectors a memòria, compara les dimensions i el valor de cada element.
- `compara_vectors_de_fitxers(char fitxer1[], char fitxer2[])`: Donats dos fitxers amb vectors escrits, compara les dimensions i que cada element coincideixi fitxer a fitxer, sense carregar-les a memòria.

- `compara_fitxers_amb_vectors(char fitxer1[], char fitxer2[])`: Donats dos fitxers, carrega a memòria els dos vectors i els compara.

Tot i haver escrit les funcions disponibles amb estructures, també s'han implementat sense. Les funcions no disponibles de la llista superior són les `carrega_matriu_array`, `carrega_vector_array`, les quals són innecessàries si no estem parlant d'estructures. Evidentment, els altres canvis són relacionats amb els retorns de cada funció, que no retornen res, però passen un punter als paràmetres que es volen omplir.

Aquesta llibreria s'usa als tests de tots els exercicis mencionats anteriorment per també exemplificar el seu ús, i els exercicis que s'han escollit per implementar-ne la solució s'han fet per a poder-los implementar amb les funcions de la llibreria.

7.7.3 Limitacions

Malauradament, CUnit ha resultat ser massa limitat per aquest projecte. A part que l'encaix de CUnit al corrector de C ha estat dificultós i ha obligat a fer sacrificis 7.7.3, ha resultat en limitacions al sistema que afecten directament a l'experiència d'usuari.

JUnit 8.1, la llibreria de tests unitaris emprada per Java, mostra quan i com la comparació d'un valor amb un altre ha fallat. Per exemple, si jo en Java defineixo dues variables amb valors diferents i les comparo amb les funcionalitats de la llibreria, al resultat de l'execució apareixerà els valors de les variables amb la frase *'Expected: value but was: other value'* tal com es pot veure a la Figura 7.35.

```
Results :

Failed tests:   test_negatius(com.example.ArrelDigitalTest): expected:<-1> but was:<-33>

Tests in error:
  test_extrems(com.example.ArrelDigitalTest): / by zero
  test_funcionament(com.example.ArrelDigitalTest): / by zero

Tests run: 4, Failures: 1, Errors: 2, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
```

Figura 7.35: Comparació mostrada quan fallen tests de JUnit. Mostra el resultat esperat però també el que ha rebut.

En canvi, CUnit no ensenya els valors de les variables que han fallat, fent que per l'usuari sigui molt més inútil el procés de correcció (tal com es veu a la Figura 7.36), ja que si li fallés un valor concret, no ho podria saber.

CUnit tampoc gestiona els *time-outs*, i s'ha hagut d'implementar amb un *script* de *bash* en execució amb la comanda *timeout*, fent la necessitat d'establir un valor relativament

```
Test: test of interpolacio_newton ...FAILED
  1. InterpolacioFuncioTest.c:30  - CU_ASSERT_DOUBLE_EQUAL(actual,expected,0.01)
  2. InterpolacioFuncioTest.c:30  - CU_ASSERT_DOUBLE_EQUAL(actual,expected,0.01)
  3. InterpolacioFuncioTest.c:30  - CU_ASSERT_DOUBLE_EQUAL(actual,expected,0.01)
  4. InterpolacioFuncioTest.c:30  - CU_ASSERT_DOUBLE_EQUAL(actual,expected,0.01)
  5. InterpolacioFuncioTest.c:30  - CU_ASSERT_DOUBLE_EQUAL(actual,expected,0.01)
  6. InterpolacioFuncioTest.c:30  - CU_ASSERT_DOUBLE_EQUAL(actual,expected,0.01)
  7. InterpolacioFuncioTest.c:30  - CU_ASSERT_DOUBLE_EQUAL(actual,expected,0.01)
  8. InterpolacioFuncioTest.c:30  - CU_ASSERT_DOUBLE_EQUAL(actual,expected,0.01)
  9. InterpolacioFuncioTest.c:30  - CU_ASSERT_DOUBLE_EQUAL(actual,expected,0.01)
  10. InterpolacioFuncioTest.c:30  - CU_ASSERT_DOUBLE_EQUAL(actual,expected,0.01)
```

Figura 7.36: Comparació mostrada quan falla un test de CUnit. No és molt explicativa.

aribtràri de segons en els quals ha d'acabar qualsevol execució de qualsevol test. Això limita certs exercicis (els que poden trigar més a obtenir el valor) i també limita l'experiència d'usuari en els exercicis més senzills i ràpids (ja que si l'alumne enviés un bucle infinit, trigaria més estona el programa a enviar l'error).

L'última limitació que trobem és la impossibilitat de què l'alumne enviï la seva implementació amb *main*, pel fet que CUnit implementa els tests en una funció *main*. Si els alumnes no envien la seva solució amb el main comentat (i les estructures que usen) la compilació del programa falla al·legant que dos main en un sol projecte són impossibles.

En resum, les limitacions principals del corrector de CUnit són:

1. Comentar el main: CUnit obliga a dir a la funció test main, perquè el compilador funcioni. Això implica, per tant, que el programa principal se li ha de comentar el main.

2. *Time out*: a l'aplicació, el time out és gestionat per un script de *bash*. Això implica haver de posar un nombre estàtic per tots els problemes, triguin més o menys. A més, el *time out* impedeix que s'executin els tests restants, és a dir, encara que només un test saltés el *time out*, farà fallar tots els altres tests.

3. Mostra dels *asserts* fallats: Per algun motiu, quan una comparació realitzada en una funció falla, no s'ensenya quin és el valor de la variable esperat ni tampoc el valor de la variable que ha fallat, donant retroacció més aviat inútil per l'usuari.

7.7.4 Criterion versus CUnit

Tal com ja s'ha comentat a l'apartat de Limitacions anterior, CUnit és un marc de proves lleuger fàcil d'utilitzar, ja que proporciona un conjunt de característiques bàsiques que suporten proves automatitzades i interactives. No obstant això, CUnit no inclou funcionalitats per executar automàticament totes les proves en una *suite*, cosa que requereix

l'addició manual de cada funció de prova a la *suite*. Aquesta característica és la que obliga a tot projecte de CUnit a contenir els tests dins d'un main.

D'altra banda, Criterion és un marc de proves unitàries més modern i avançat. Inclou suport per al registre automàtic de proves, diferents tipus d'afirmacions i característiques avançades. Un aspecte a destacar de Criterion és el seu suport integrat per al control de senyals, que li permet continuar executant altres proves després que una hagi fallat o hagi causat un comportament indefinit. A més, Criterion permet executar proves de manera aïllada en processos separats, el que ajuda a evitar que els efectes secundaris d'una prova afectin altres proves. Criterion també pot produir informes detallats en diversos formats, cosa que resulta útil en un entorn d'integració contínua.

Pel que fa als escenaris específics de divisió per zero i bucles infinits que hem mencionat a la secció 7.7.3, els dos marcs de proves tenen diferents comportaments. En CUnit, una divisió per zero en una prova normalment farà que tot el procés de proves es bloquegi, ja que CUnit no proporciona cap maneig integrat per a aquest escenari. En contrast, Criterion executa cada prova en un procés separat. Si es produeix una divisió per zero, això causarà un senyal *SIGFPE* i matarà el procés del test concret, però Criterion en capturarà els resultats i informarà que la prova ha fallat, continuant amb les proves restants.

Tocant l'altre punt feble de CUnit, la gestió dels bucles infinits, CUnit roman a la espera indefinidament, haven de posar un intermediari en Bash a l'execució per controlar el cas de `TIME_OUT`. En canvi, en Criterion, per la gestió de subprocessos i senyals, quan una prova entra en un bucle infinit pot matar manualment el test concret sense afectar ni les següents proves ni el sistema en si. Actualment, Criterion no té funcionalitat de temps d'espera integrada, però hi ha maneres d'afegir límits de temps manualment si és necessari. Per tant, si només un dels tests entra en un bucle infinit (tot i haver-se d'afegir la gestió a mà) els altres tests executaran, no com amb CUnit.

Per acabar, Criterion ensenya el valor de les variables quan fallen, donant molta més informació als usuaris quan un test dona el resultat esperat.

En resum, Criterion és més avançat i més resistent contra proves amb comportaments inesperats, té més característiques i una interfície més moderna. Cosa que el fa un gran candidat per substituir CUnit en aquest projecte. Malauradament, pel marc temporal inherent en aquest projecte, no s'han pogut traduir tots els tests a Criterion ni tampoc implementar la llibreria descrita als apartats anteriors amb aquest. Així i tot, el projecte es beneficiaria gratament de la seva inclusió (vegis secció 10).

Capítol 8

Implementació

A la següent secció descrivim les tecnologies emprades en el desenvolupament d'aquest projecte, així com els requisits tècnics necessaris per al codi. Després, s'analitzen en detall diverses parts tècnicament complexes del projecte i s'explica el seu funcionament.

8.1 Tecnologies Emprades

Python

Python és un llenguatge de programació interpretat mutliparadigma de propòsit general d'alt nivell. És un llenguatge amb un lèxic molt llegible i la gran quantitat de llibreries que conté el fan molt adequat per una gran varietat de projectes. Concretament en aquest projecte s'han usat les següents llibreries:

- `python-telegram-bot`: implementa molts dels protocols per comunicar-se amb Telegram i classes que abstrauen tota la informació necessària per comunicar-se amb l'usuari. També conté gran quantitat de documentació i exemples que s'han emprat com a base d'aquest projecte.
- `gitpy2`: és una llibreria tècnica de baix nivell de git. Permet clonar i fer push de repositoris, i els elements que proporciona han permès implementar un equivalent de les commandes de *git hard reset* i *git pull* d'un repositori.
- `docker SDK for Python`: fa abstraccions sobre tota la llibreria de Docker, des de construir contenidors, imatges, executar aquestes, crear volums i molt més. També disposa de mètodes per obtenir els logs dels contenidors, i altres crides directes a l'API de Docker, que poden ser funcionals a molts nivells.

- `github-webhooks-framework`: llibreria que implementa diverses classes per rebre webhooks de github. Disposa de molta documentació, de la qual s'ha basat l'aplicació `manager_db` d'aquest projecte.
- `mariaDB`: permet connectar-se amb MariaDB des de Python.
- `AsyncIO`: llibreria per permetre codi concurrent amb fil-únic utilitzant corutines, multiplexant l'accés d'I/O a través de *sockets*, distribuir tasques en cues i altres recursos, permetent gestionar concurrència en servidors web. S'ha emprat en tots els mòduls de servidors d'aquesta aplicació.

FastAPI

FastAPI és un framework web per construir APIs amb versions superiors de Pyhton3.7. Gràcies a la seva implementació d'Starlette y Pydantic, és un dels marcs de Python més ràpids, combinant la facilitat pel programador amb un rendiment excel·lent, complementat amb una documentació moderna per poder acomplir els projectes que el programador té presents. Starlette ofereix gestió asíncrona interna dels processos d'alt rendiment i Pydantic validació d'estructures i dades d'*endpoints* mitjançant un sistema de tipus. Tots els servidors de tots els mòduls (`GestorDB`, `Xatbot` i `Corrector`) d'aquest projecte estan escrits en FastAPI, concretament `Xatbot` també usa Starlette com a complement.

Gunicorn

Gunicorn és un servidor HTTP WSGI per Python, conegut també com a "Green Unicorn". Es tracta d'un servidor web lleuger i altament configurable que s'utilitza principalment per servir aplicacions web Python en producció. Gunicorn es basa en l'especificació *Web Server Gateway Interface* (WSGI) que estandarditza la interacció entre els servidors web i les aplicacions web Python. Permet als desenvolupadors desplegar les seves aplicacions web de manera ràpida i fàcil, sense haver de preocupar-se per les complexitats de la manipulació directa de sol·licituds HTTP. Aquest servidor web és particularment útil quan es treballa amb aplicacions basades en marcs com FastAPI, ja que pot gestionar múltiples sol·licituds simultàniament i pot ser ajustat per aconseguir un rendiment òptim en diferents entorns.

En el context d'aquest projecte, Gunicorn es fa servir com a servidor de producció per a les aplicacions desenvolupades amb FastAPI. Aquesta combinació permet gestionar eficientment les sol·licituds HTTP, mantenir una alta disponibilitat i garantir un rendiment robust en el desplegament de l'aplicació.

Docker

Docker és un projecte de codi obert que automatitza el desplegament d'aplicacions dins de contenidors de programari, proporcionant una petita màquina virtual que consumeix pocs recursos a on executar el codi desitjat. Aquestes aplicacions es poden construir de zero o sobre imatges d'altres Docker que ja et garanteixen implementacions de diverses funcionalitats. Docker s'usa per empaquetar una aplicació i totes les seves dependències a un contenidor virtual que serà executable a qualsevol entorn Linux, fet que la fa increïblement versàtil i adequada per desplegar l'aplicació a servidors. En aquest projecte s'han usat imatges de Docker per tots els correctors i pels mòduls de l'aplicació `GestorDB`, `Xatbot`, `MariaDB` i `RASA`, ja que és una manera senzilla d'empaquetar totes les dependències necessàries i per desplegar tres aplicacions arribat el moment de producció.

Docker Compose

Docker-Compose és una eina per definir i executar aplicacions multidocker. Des d'un fitxer YAML a l'arrel de l'aplicació, s'especifiquen les característiques de cada Docker, com nom, tipus de build, prioritat d'execució i ordre d'engegada, igual que múltiples característiques d'intercomunicació entre els Dockers, permentent obrir ports que es comuniquen mitjançant una xarxa virtual creada per Docker Compose. Amb Docker Compose pots fàcilment encendre, apagar o reconstruir l'aplicació, comprovar l'estat del servei, comprovar els logs i executar comandes de manera externa. Per exemple, en aquest projecte s'usa Docker-Compose per gestionar l'estat de quatre mòduls dockeritzats: `GestorDB`, `Xatbot`, *MariaDB* i *Rasa* i *Rasa Action Server* tal com es veu a la Figura 8.1. Si no el féssim servir, hauríem d'encendre cadascun d'ells per separat i comprovar de manera externa que estan operatius i són capaços de rebre peticions. Amb Docker-Compose, només és una comanda per encendre, gestionant ell la comunicació amb xarxes virtuals, *healthcheks* i comprovacions de si el servei està disponible fàcilment indicades pel programador.

Telegram

Telegram és una plataforma de missatgeria instantània que destaca per la seva seguretat robusta i funcionalitats riques. Aquesta plataforma ofereix APIs públiques que permeten als desenvolupadors interactuar amb els usuaris a través de bots, proporcionant una interfície de comunicació dinàmica. Aquests bots de Telegram són programes autònoms que actuen com a intermediaris entre l'usuari i la plataforma, permetent el desplegament de funcionalitats específiques o aplicacions, inclosa la capacitat de respondre a peticions d'usuaris, enviar notificacions automàtiques, gestionar grups, i més. En aquest projecte, Telegram ha estat utilitzat per facilitar la interacció entre l'usuari i l'aplicació a través de l'ús d'un bot de Telegram implementat en Python. Aquesta integració de Telegram

permet un flux d'interacció ràpid i eficient, facilitant la usabilitat general de l'aplicació.

MariaDB

MariaDB és un derivat dels sistemes de gestions de bases de dades MySQL impulsada per la comunitat per mantenir el seu estatus lliure sota la llicència GNU GPL, creada en el moment de l'adquisició de MySQL per part d'Oracle. A causa d'això, manté una gran similitud amb MySQL i molta compatibilitat amb totes les tecnologies ja integrades, com també dissenys de la comunitat de tecnologies noves que requereixen llicència d'ús. Al ser totalment lliure, hi ha una gran quantitat d'informació i suport en línia, com també una imatge de Docker de la mateixa totalment compatible amb Docker-Compose, fet que ha equilibrat la balança a usar MariaDB en comptes de MySQLite o similars.

Maven

Maven una eina de programari per la gestió y construcció de projectes Java basada en XML. Utilitza un *Project Oriented Model* (un fitxer POM.xml) per a descriure el projecte de programari a construir, incloent-hi els mòduls necessaris (java en el nostre cas) i components externes per l'execució d'aquest (llibreries de testeig i dependències de Maven) juntament amb quin ordre d'execució ha de seguir. Ja conté diverses eines per tasques predefinides, com la compilació i execució de codi Java, implementant-lo en el corrector de Java en aquest projecte.

Java

Java és el llenguatge de programació interpretat orientat a objectes per excel·lència amb llicència GPL de codi obert. És un llenguatge compilat amb una màquina virtual d'intermediari que permet una reutilització de codi molt elevada (permetent trobar-hi una enorme quantitat de llibreries ja implementades) a canvi d'una lleugera reducció al rendiment en comparació al llenguatge del qual va ser dissenyat, C. És un llenguatge flexible i potent, molt emprat i valorat en el desenvolupament d'aplicacions distribuïdes pel fet que la màquina virtual el faci, per defecte, un llenguatge multiplataforma.

JUnit

JUnit és una llibreria de proves unitàries per al llenguatge de programació Java que desenvolupa un paper clau en el desenvolupament d'aplicacions robustes i fiables. És una eina essencial per a la realització de tests unitaris, facilitant la detecció precoç i sistemàtica

d'errors durant el desenvolupament del codi. JUnit proporciona una API senzilla per a la creació de proves i la comprovació de resultats, facilitant així la implementació de procediments de proves automatitzats. En aquest projecte, JUnit s'ha emprat per implementar els tests dels exercicis de Java, que s'executen amb el codi fet per l'alumne, podent-li donar retroaccions sense l'assistència del docent de l'assignatura.

C

C és un llenguatge fortament tipat de nivell mitjà (per les estructures) però amb moltes característiques de baix nivell, com la gestió de la memòria dinàmica per part del programador. És el llenguatge de programació més eficient i veloç, sobretot apreciat pel desenvolupament d'aplicacions de programari de sistemes. Es va crear per suplir la necessitat d'un llenguatge molt més flexible (i intuïtiu/fàcil de llegir) que l'assemblador, però que mantingués la característica de ser un llenguatge molt proper a la màquina. C aconsegueix velocitats iguals al llenguatge màquina amb moltes menys instruccions pel programador, cosa que li dona el privilegi de ser el llenguatge en el qual està escrit el nucli del sistema operatiu Unix.

Makefile

Makefile és una eina que permet gestionar i automatitzar el procés de compilació de codi en llenguatges de programació. Permet definir un conjunt de regles que especifica com es construeixen les dependències entre els fitxers de codi i com es generen els executables o llibreries del projecte. Per tant, l'ús de Makefile facilita l'execució del codi quan el projecte es faria massa gran, permetent automatitzar tasques com la compilació, la vinculació, l'execució de tests o la generació de documentació. També proporciona mecanismes per a la recompilació selectiva, de manera que només els fitxers que han estat modificats són recompilats, millorant així l'eficiència del procés de desenvolupament. En aquest projecte, Makefile s'ha utilitzat per gestionar la compilació i l'execució del codi de la implementació dels exercicis dels alumnes en C dins de l'entorn de Docker.

Valgrind

Valgrind és una eina d'anàlisi de programari que permet detectar problemes de memòria i de concurrència. Es tracta d'un conjunt d'eines per a la depuració i la creació de perfils de programari que ofereix diverses funcionalitats per ajudar a identificar errors com fuites de memòria, desbordament de buffers, ús de memòria no inicialitzada, etc.

Valgrind s'executa sobre binaris compilats, de manera que no és necessari modificar o recompilar el codi per usar-lo. Això fa que sigui una eina flexible i poderosa, que pot ser

feta servir per millorar la qualitat del codi en qualsevol projecte que ampri C.

L'ús de Valgrind es fa particularment important en els projectes on la gestió de la memòria és crítica, com és el cas del codi escrit en C, on el programador té la responsabilitat de gestionar explícitament la memòria dinàmica. En el context d'aquest projecte, Valgrind ha estat fet ús per depurar i assegurar la correcta gestió de la memòria en la implementació dels exercicis dels alumnes en C. En aquest projecte s'usa per comprovar la gestió dels alumnes de la memòria dinàmica i puntuar-los al respecte.

CUnit

CUnit és una llibreria que permet implementar tests unitaris per a aplicacions en llenguatge C. Aquesta llibreria de testatge ofereix un entorn senzill i eficient per a la creació, execució i anàlisi de casos de prova específics per a funcions o mètodes del codi en C. CUnit facilita la identificació de problemes i errors en les funcions implementades, permetent als desenvolupadors millorar la qualitat del codi i assegurar-se que els mètodes implementats compleixin amb les expectatives de rendiment i precisió. La seva integració en projectes és senzilla, i disposa d'una àmplia documentació i suport de la comunitat per a facilitar-ne l'ús i l'adopció en qualsevol projecte que utilitzi el llenguatge C. En el context d'aquest projecte, CUnit es va utilitzar per implementar una llibreria de proves aplicades a les assignatures de Mètodes Numèrics u i dos, contenint funcions i de comparació de Matrius (vectors de doble punter) i Vectors (arrays simples).

GitHub

GitHub és una plataforma de control de versions i col·laboració per desenvolupadors de software. La plataforma està basada en Git, un sistema de control de codi de lliure distribució que pretenia millorar la creació i desenvolupament de programari per agilitzar-lo. S'utilitza per emmagatzemar el codi font de les aplicacions, monitoritzar els canvis i mantenir-los en històrics. També facilita la cooperació entre programadors de manera eficient amb eines per gestionar possibles conflictes abans de penjar el codi font, donant eines als programadors per no sobreescriue en cap moment els canvis d'altres. En aquest projecte el codi dels exercicis que l'agent conversacional pot lliurar (fitxers contenint l'enunciat i les definicions del mètodes/funcions a implementar) s'emmagatzemen en repositoris privats on tenen accés els docents.

Rasa Open Source

Rasa és una llibreria open-source que proporciona eines per crear xatbots i assistents virtuals amb capacitats d'enteniment del llenguatge natural (NLU, Natural Language Un-

derstanding). Rasa està dissenyat per a construir assistents que entenen les consultes dels usuaris en llenguatge natural i responen adequadament. La plataforma utilitza tècniques d'aprenentatge automàtic i profund per comprendre i processar les consultes dels usuaris, permetent als desenvolupadors de l'agent crear respostes personalitzades. Rasa és independent de la plataforma i pot ser integrat amb diferents canals de missatgeria com Slack, Telegram, o la web. En aquest projecte, Rasa s'ha usat per implementar un agent conversacional capaç d'interactuar amb els estudiants, respondre a les seves preguntes i proporcionar-los enunciats i codis base per a diferents exercicis de programació.

GPT3-5-turbo

GPT3-5-turbo és un model de llenguatge generatiu avançat desenvolupat per OpenAI. Basat en l'arquitectura GPT-3, aquest model està dissenyat per comprendre i generar text en llenguatge natural a partir de *prompts* (peticions) proporcionats. Amb una enorme quantitat de paràmetres, GPT3-5-turbo és capaç de generar respostes de qualitat humana a una àmplia varietat de consultes, fent-lo una eina extremadament útil en una gran quantitat d'aplicacions, incloent-hi bots de conversa, assistents virtuals, generació automàtica de contingut i més.

GPT3-5-turbo pot funcionar de forma autònoma o pot ser entrenat en tasques específiques per millorar la seva precisió i utilitat en contextos específics. Tot i que la implementació d'aquest model requereix una comprensió profunda de les tècniques d'aprenentatge profund, la seva potència i flexibilitat el fan una eina valuosa per a molts projectes de desenvolupament de programari.

En el context d'aquest projecte, GPT3-5-turbo s'ha utilitzat per implementar la capacitat de l'agent conversacional per respondre a preguntes i consultes en temps real dels estudiants. Aquest model ha proporcionat a l'agent la capacitat de comprendre les consultes en llenguatge natural dels estudiants, generar respostes apropiades i proporcionar-los amb informació i recursos útils per al seu aprenentatge.

Per conèixer les versions i programari necessari extra per executar el codi, vegis l'apèndix manual tècnic del projecte.

8.1.1 Tecnologies Descartades

Els dos projectes originals [6] [5] empraven com a servidor web de les aplicacions Flask. A la reimplementació del projecte es va decidir usar FastAPI perquè se sabia amb seguretat que era capaç de gestionar les crides concurrents i l'asincronia i a la documentació de *python-telegram-bot* i de *github-webhook-frameworks*, que usaven FastAPI. També la decisió és per l'experiència de l'autor d'aquest projecte, ja que havia fet servir FastAPI abans.

Pel corrector de Java també es va considerar Gradle, una eina exclusivament de tests (no com Maven que et permet executar projectes sencers) però no vam aconseguir construir un Docker amb Gradle que funcionés.

Heroku va ser considerada al principi del projecte pel deploy i manteniment de l'aplicació, però després de diversos intents infructuosos, es va decidir optar per AWS, que permetia tenir diverses execucions de Docker simultànies i ens permetia circumnavegar les limitacions d'Heroku amb els contenidors.

Pel corrector de C es van considerar diverses llibreries de correcció com GTest, cmocka i Crietrion. Com que cap era GTest i Criterion eren també per C++, no es van escollir. Entre CUnit i cmocka, semblava que CUnit tenia més documentació i una estructura de projecte més senzilla, per això cmocka va ser descartada.

8.2 Diagrama d'Implementació

En aquest apartat mostrem el diagrama de programari corresponent a l'arquitectura real de l'aplicació, no a les estructures abstractes de les quals ens hem servit a disseny per explicar el seu funcionament. Aquest diagrama es troba a la Figura 8.1.

Tal com es veu a la Figura, el canvi principal respecte el diagrama de programari de disseny general de la Figura 7.1, és que el mòdul `Xatbot` està separat. Això és perquè *Rasa* i el *Rasa Action Server* operen com a servidors, no com a mòduls, per tant, s'han d'activar com un servidor normal. Això ens dona un total de cinc servidors en aquest projecte, els tres discutits a l'apartat de disseny (`Xatbot`, `Corrector` i `GestorDB`) i els dos de *Rasa* que just acabem de mencionar. Així i tot, a disseny s'han ajuntat, ja que tot *Rasa* actua supeditat al mòdul de `Xatbot`, és a dir, no li arriba directament cap petició i tampoc en pot fer a ningú més que a `Xatbot`.

Entre els mòduls s'han establert regles de comunicació sobre que ha de retornar cada mòdul depenent de l'acció realitzada. Totes, davant d'una petició, retornen com a resposta un JSON, una miqueta diferent depenent del cas.

- Si s'afegeix un ítem a la base de dades, {`"status":'added'`} si ha estat exitós, {`"status":'not_added'`} si no s'ha pogut fer, però el programa no ha retornat excepció.
- Si es vol comprovar si un ítem és a la base de dades, {`"status":'found'`} si s'ha trobat, {`"status":'not_found'`} si no.
- Si es vol actualitzar un valor de la base de dades, {`"status":'updated'`} si s'ha actualitzat, {"status":'not_updated'} si no.

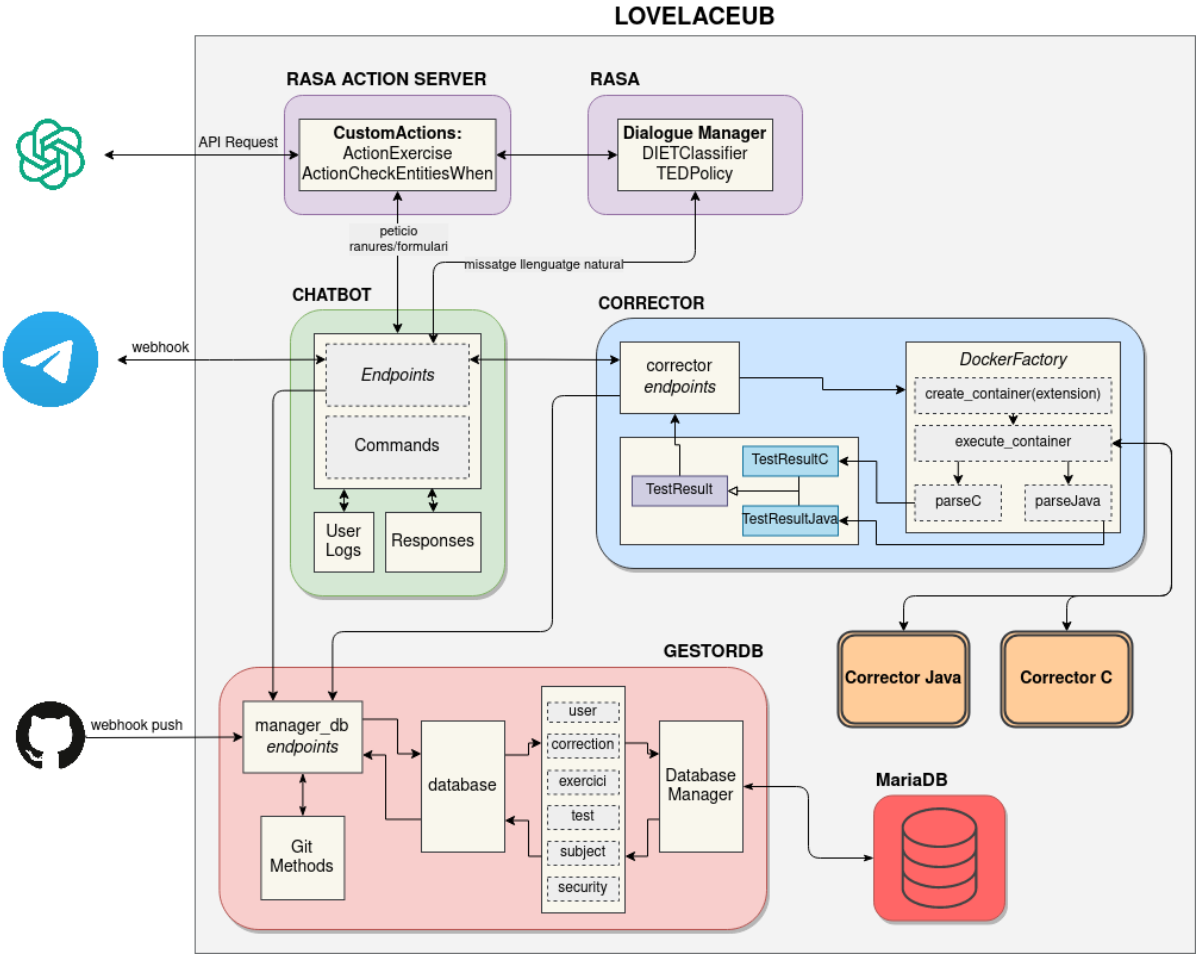


Figura 8.1: Diagrama de software en detall de la implementació de l'agent conversacional LovelaceUB

- Resposta a la petició d'un fitxer tant d'exercici com test: {`"status":'found', 'content': bytes, 'filename': nom`}
- Comunicació amb RASA: {`'user_id': id, 'text': missatge`}
- Recepció de petició de Telegram a Rasa de si s'ha enviat el fitxer a través del xat: {`"status":'sent'`} i {`"status":'not_sent'`}
- Enviament d'un missatge a Rasa: {`ïd": user_id, "message": text`}

En cas que alguna de les comunicacions hagi llançat algun error, els *endpoints* retornen sempre {`"status":"error", "message": detalls`}.

8.2.1 Mòduls Dockeritzats

En tenir el projecte cinc aplicacions que s'han d'aixecar cada vegada que s'ha d'executar l'aplicació, s'ha necessitat una tecnologia que permetés fer-ho de manera relativament senzilla. L'ús de contenidors de Docker amb el sistema Docker-Compose permet gestionar diverses imatges i construir múltiples contenidors de diferents mòduls amb només una comanda. Aquest procés es controla mitjançant un fitxer YAML amb tots els paràmetres necessaris contemplats per l'execució correcta.

Prenent de referència la Figura 8.1, es troba que tota aplicació amb servidor ha estat *dockeritzada* excepte una. Les *dockeritzades* amb imatges modificades són `Xatbot` i `GestorDB`, mentre que les usades amb imatges prefetes són *MariaDB*, *RasaNLU* i *Rasa Action Server*. `Xatbot`, `GestorDB` i *Rasa Action Servers* es fan amb Dockerfiles escrits per aquest projecte basant-se en imatges de Python per les dues primeres i la de Rasa pel l'*Action Server*. Rasa fa servir la imatge prefeta de la comunitat de Docker, fet que la fa molt més segura i eficient igual que *MariaDB*. El mòdul no aixecat amb Docker-Compose és `Corrector`, fet que té a veure amb la seguretat de l'aplicació, la justificació de la d'aquesta es pot trobar a més endavant a l'apartat 8.7.3

8.2.2 Avantatges de Docker-Compose

Més enllà de els avantatges evidents de la gestió de múltiples imatges amb només una comanda, Docker-Compose permet implementar diverses funcionalitats al YML útils de per si, que nosaltres emprem a l'aplicació per garantir-ne el correcte funcionament:

- Càrrega de variables d'entorn: Totes les variables associades i necessàries per al funcionament de les aplicacions es carreguen a l'entorn de Docker-Compose, evitant la necessitat que cada mòdul les carregui a memòria individualment.
- *Healthcheks* i *depends on*: Hem implementat la següent jerarquia en l'aixecament d'aplicacions per assegurar-nos que el projecte està *healthy* (és a dir, està preparat per rebre peticions) abans de començar a fer peticions:
  - Des de `GestorDB` a `MariaDB`: ping per comprovar que la base de dades està operativa
  - Des de `Xatbot` a `GestorDB`: curl a l'endpoint *heathckeck* de `GestorDB`, que comprova si `Corrector` rep peticions i també si la connexió a la base de dades està operativa.
- Xarxes Virtuals: Docker-Compose permet crear xarxes virtuals entre els contenidors per adreçar-s'hi a través del nom de les imatges i no de la IP (és a dir, `http://chatbot:8443/endpoint` en comptes de `http://0.0.0.0:8443/endpoint`)

que facilita molt la comunicació entre imatges. Per la connexió a un contenidor des d'una aplicació no *dockerizada*, Docker-Compose permet mantenir fixa una IP dins de la xarxa, per fer que l'atac a aquesta des de fora sigui a una IP concreta no canviant. En aquest projecte hem definit IP estàtiques pels mòduls `Xatbot` i `GestorDB` per facilitar la comunicació d'aquests amb el `Corrector`.

- Exposició de ports interns diferents dels externs: Docker fa distinció dels ports oberts a la màquina local que els oberts a la teva xarxa virtual, és a dir, que potser Docker accepta peticions del port 8000 de la màquina local, però les pot redirigir al port 5000 de la xarxa virtual. Això permet tenir més flexibilitat en els ports que s'obren i, per exemple, posar les aplicacions en ports contigües mentre que des de la màquina local s'ataquen altres ports.

Tot i els avantatges mencionats de proporciona l'ús de Docker i Docker-Compose, un factor a tenir en compte és la primera construcció d'un contenidor, on Docker ha de descarregar-se les dependències i les llibreries d'execució, procés que pot ser molt llarg, depent del tamany de les dependències. Per exemple, la imatge de Docker de Maven és molt gran, necessita diversos minuts per a baixar-se depenent de la connexió a internet disponible. Si per cada correcció l'agent es demorés tant, l'experiència d'usuari seria horrible per l'alumne.

Per millorar l'experiència de l'alumne s'ha emprat un Docker multiestadi. Un Docker és multiestadi si es construeix per part, en el cas del corrector de Java implementa dos estadis:

- La primera part és la baixada de les dependències i la construcció del projecte de Maven. En aquest estadi no es copien els fitxers a corregir de l'usuari.
- Es copia el *pom.xml*[1], els fitxers i tests a executar i s'executen.

L'avantatge del multiestadi és que si Docker no detecta canvis en el Dockerfile, reutilitza la construcció de la primera part que ja té a memòria. És a dir, si ja ens hem baixat les dependències un cop i el Dockerfile o el projecte no han estat modificats, aquest no es construeix i es reutilitza el de l'anterior execució. Només es copien els fitxers de la nova correcció i s'executen, quasi arreglant el problema de l'experiència de l'usuari. Diem quasi perquè el que no soluciona la construcció multiestadi és que les dependències s'han de descarregar un cop en algun moment, i que el primer usuari que intenti corregir un exercici s'hagi d'esperar el temps de les dependències segueix sent inacceptable pel que fa a l'experiència d'usuari. Per arreglar-ho, s'ha implementat *Dummy Tests* per Java i un per C, els quals consisteixen a testejar el mètode/funció "return 1". Quan s'aixeca el corrector executa aquests dos tests, i fins que no s'han acabat les dues correccions, el

[1]Project Oriented Model: fitxer xml que defineix diverses característiques del projecte de Maven com els directoris, la versió de java, el package d'execució...

corrector no accepta peticions. Així el primer usuari no s'ha d'esperar per la primera correcció, ni els programadors hauran d'estar atents en el moment de desplegar l'aplicació o quan caigui per qualsevol motiu.

8.3 Peticions a Xatbot

Aquesta secció es tractarà el què implica usar *webhooks* en el desenvolupament d'un servidor web: començant pels sistemes de gestió de múltiples peticions externes, seguint amb la llibreria AsyncIO i les seves capacitats i com s'ha acoblat Telegram a aquesta arquitectura

8.3.1 *Webhooks* i Asincronia en Servidors Web

La gran majoria de les comunicacions d'internet es basen en el model client-servidor, on el client envia una petició a un servidor perquè el segon realitzi quelcom i li enviï la resposta. Es troben dues grans maneres de dur a terme aquest processament: de manera síncrona o de manera asíncrona.

Fent-ho síncronament, el client fa la petició al servidor i espera fins que rep la resposta del servidor; aquest procés s'anomena bloqueig (o *blocking*), ja que el client no pot executar cap altra tasca mentre espera. En canvi, fent el procés asíncronament el client no roman esperant la resposta del servidor i continua fent altres tasques; aquest procés s'anomena desbloqueig (o *non-blocking*), i permet al client fer més tasques donat un interval de temps.

Per tant, l'asincronia fa el sistema més eficient, però s'aconsegueix sacrificant la simplicitat de la comunicació síncrona. S'ha d'introduir un gestor de peticions que pugui emmagatzemar les tasques just quan arribin, proporcionar-les al servidor de mica en mica per no sobrecarregar-lo i que avisi al client correcte en rebre la resposta[8]

Els *webhooks* són un tipus de processament asíncron i, per tant, s'ha d'implementar un sistema de gestió entre el client i el servidor per evitar problemes. Concretament, els *webhooks* poden ser considerats esdeveniments[10], en conseqüència un bon gestior és implementar una cua, que és el que fa el mòdul `Xatbot` i pel que l'exemple de *python-telegram-bot*[14] va ser escollit com a base per a reimplementar el projecte des de zero, de la mateixa manera amb `GestorDB` i l'exemple de *github-webhooks-framework*[7]. Això també segueix l'argument presentat a la secció de Projectes Relacionats 4 sobre tornar a començar el projecte de zero: el codi dels projectes anteriors implementaven la comunicació amb Telegram i GitHub mitjançant *webhooks*, però no els gestionaven asíncronament. Aquest fet hauria pogut portar a alguna de les següents malfuncions [9]:

- Control de la velocitat de recepció: Si el servidor rep molts *webhooks* simultàniament, pot acabar sobrepassat, tant per l'intent d'executar massa processos per part del servidor o perquè el client ha tingut una acumulació de *webhooks* per enviar.
- Gestió d'errors: Hi ha molts factors que poden causar errors es tracta amb *webhooks*. Això inclou recursos insuficients, dependències d'altres serveis que no estan funcionant correctament o problemes amb el codi del servidor.
- Gestió d'interrupcions de servei: Les caigudes del servei són inevitables en sistemes informàtics. Cal preparar-s'hi i assegurar-se que el sistema pot continuar lliurant resultats consistents, encara que hi hagi hagut una interrupció de servei.
- Gestió de temps d'espera: Algunes plataformes tenen un límit de temps per processar un *webhook*. Si el teu servidor no aconsegueix processar-lo dins d'aquest temps, la plataforma pot considerar que ha fallat, encara que acabi processant-lo correctament més tard.

La implementació de la cua és molt similar a les dues aplicacions: quan Telegram o GitHub disparen el *webhook* (clients), les aplicacions reben la petició, l'emmagatzemen en una cua prioritària i el servidor les va processant al seu ritme. Un cop el servidor ha acabat, reenvia la resposta al client adequat.

En canvi, els avantatges de quan s'implementen asíncronament de manera correcta[9]:

- Millores en l'escalabilitat: Un dels majors avantatges dels *webhooks* asíncrons és la seva capacitat per manejar un gran nombre de peticions de forma eficaç. En lloc d'haver de processar cada sol·licitud immediatament i una per una (com en un model síncron), un model asíncron pot encolar les peticions i processar-les quan els recursos ho permetin. Això pot ajudar a prevenir situacions en què el sistema es veu desbordat per un gran nombre de peticions.
- Tolerància als errors i a les interrupcions de servei: En un model asíncron, si una petició falla per qualsevol motiu (com un error d'execució o una interrupció del servei), pot ser reintentada automàticament sense afectar altres peticions, ans al contrari que el model síncron, on un error pot provocar el fracàs de tota la seqüència de peticions.
- Temps de resposta més ràpids: Un altre avantatge dels *webhooks* asíncrons és que poden proporcionar temps de resposta més ràpids. Com que les peticions es poden processar en paral·lel, l'usuari no ha d'esperar que una petició es completi abans que la següent pugui començar.
- Maneig eficient dels temps d'espera: En un model asíncron, no és rellevant el que triga a processar-se una petició. Si trigués més de l'esperat, no causaria error de connexió, ja que les peticions es processen independentment l'una de l'altra.

- Més control sobre el flux de treball: Un model asíncron permet tenir més control sobre el flux de treball. Prioritats poden ser establertes per a diferents tipus de peticions, decidir quines peticions es poden processen primer i reorganitzar la cua segons les necessitats del sistema.

8.3.2 Concurrència per Subrutines amb AsyncIO

En la programació en paral·lel o concurrent, *AsyncIO* és una llibreria de Python que implementa la gestió de tasques requerint espera (per exemple les peticions de xarxa) sense bloquejar tot el programa. En el context d'aquest projecte, s'ha aplicat aquesta llibreria al llarg dels mòduls `Xatbot` i `GestorDB` per a gestionar les peticions de *webhooks* provinents de Telegram i GitHub.

Amb l'ús d'*AsyncIO*, quan el servidor rep una petició, no es bloqueja fins que aquesta està completament processada. En comptes d'això, la petició es delega a una tasca en segon pla, permetent que el servidor en continuï atenent d'altres. Això és particularment beneficiós quan es tracta amb peticions que poden tardar una quantitat significativa de temps per processar-se, o per la seva complexitat computacional o per problemes tècnics, ja que permet que el servidor segueix atenent mentre el procés feixuc roman en segon pla.

Aquesta arquitectura concurrent no solament incrementa l'eficiència del servidor, sinó que té el potencial d'augmentar de manera substancial la qualitat de l'experiència d'usuari. Les sol·licituds poden ser processades de forma més ràpida, disminuint així el temps d'espera percebut. És crucial comprendre, tanmateix, que la concurrència facilitada per AsyncIO no es correspon amb la paral·lelització que es pot obtenir mitjançant l'ús de múltiples fils o processos.

Tot i que la documentació oficial de Python caracteritza *AsyncIO* com una biblioteca destinada a la programació concurrent, aquesta no se sustenta en l'ús de múltiples fils o processos. En realitat, *AsyncIO* és una implementació que opera en un disseny d'un sol fil, un sol procés, fent ús de la multiprogramació cooperativa, que procedirem a definir a continuació. En altres termes, *AsyncIO* proporciona una semblança de concurrència a pesar de l'ús d'un únic fil en un únic procés. Les corutines, un element central d'*AsyncIO*, poden ser programades de manera concurrent, però no són inherentment concurrents.

Per assentar aquesta idea, *AsyncIO* representa un estil de programació concurrent, però no es tracta de paral·lelisme. Està més alineada amb la idea de fils que amb la de múltiples processos, però difereix significativament de tots dos i constitueix una eina independent en el catàleg d'instruments per a la concurrència.

Aquesta explicació ens porta a un altre concepte. Què significa que alguna cosa sigui asíncrona? Encara que no és una definició exhaustiva, per als nostres propòsits en aquest context, podem considerar dues propietats:

- Les rutines asíncrones poden aturar mentre esperen el resultat final i deixar córrer altres rutines.
- El codi asíncron, a través del mecanisme prèviament descrit, facilita l'execució concurrent. D'altra manera, el codi asíncron confereix l'aparença i l'experiència de la concurrència.

Per tant, malgrat que AsyncIO i la concurrència que aquesta suporta no està directament associat amb el concepte de *webhooks*, aquests faciliten la gestió eficaç d'aquestes sol·licituds en un entorn en temps real. Un servidor adequadament implementat amb *AsyncIO* pot administrar de manera més eficient una gran quantitat de sol·licituds de *webhooks*, millorant així el rendiment global del sistema.

8.3.3 Telegram

La llibreria *python-telegram-bot* permet associar l'execució d'una funció quan una comanda és enviada per l'usuari a Telegram. A continuació es mostra el diagrama de seqüència de les comandes entra i registra i què realitza cada funció segons el nombre d'arguments (quantes paraules separades per espais hi ha després de la comanda `/comanda argument1 argument2`) que l'usuari posa.

Telegram és la interfície per l'usuari a connectar-se al projecte. Per connectar el bot amb el xat de Telegram, es construeix el bot amb el TOKEN de Telegram, això crea una abstracció del xat amb l'usuari on es fan peticions. Per enllaçar el bot amb el *webhook* s'usa *python-telegram-bot* on es posa la direcció del servidor juntament amb l'*endpoint* `"/telegram"`, així quan un usuari enviï un missatge s'atacarà la recepció preparada per aquesta gestió.

Cada usuari a Telegram té associats dos paràmetres d'identificació, la *user_id* i el *chat_id*. Aquests són intercanviables en la gran majoria de contextos, però no identifiquen el mateix [5]. En aquest projecte, a l'implementar converses un a un, s'ha escollit fer servir la *user_id* per totes les interaccions.

Tal com hem mencionat a la secció 7.2.1, s'han implementat diverses comandes per facilitar la interacció de l'usuari amb l'agent.

8.4 Xatbot i Rasa

Aquesta secció tracta de l'acoblament de l'arquitectura existent controlant el xat de Telegram i les comandes i el gestor d'NLU de Rasa: es comença explicant l'estructura d'un projecte Rasa, com s'ha conectat amb l'arquitectura existent i acabant a on i com es guarden els *logs* d'usuari.

8.4.1 Estructura del Projecte de Rasa

Tal com s'ha mostrat al diagrama d'implementació, Rasa necessita dos servidors dedicats per funcionar, el NLU i l'Action Server. L'NLU és l'encarregat de detectar de comprendre el llenguatge natural tal com s'ha explicat al llarg d'aquest projecte, i l'Action Server executa les accions que hem descrit al llarg del codi. Ara s'explicarà de com està estructurat un projecte de Rasa, tal com es mostra a la Figura 8.2.

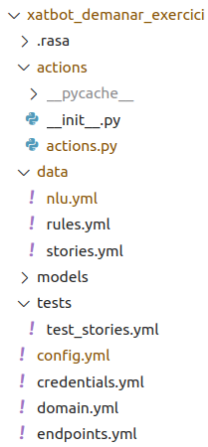


Figura 8.2: Estructura del projecte de Rasa

Expliquem ara el format del projecte vist a la Figura 8.2:

El format YAML[2] és un format de dades llegible per humans i àmpliament utilitzat per a la configuració de projectes de programari.

domain.yml: Aquest és el fitxer principal de definició del projecte. Defineix el domini de l'aplicació de Rasa, és a dir, l'entorn en què l'agent es desenvolupa. Conté la definició de les intencions que pot dur a terme l'usuari, les entitats a extreure dels missatges, les ranures (variables que permeten a l'agent recordar informació específica), formularis (recull de dades de l'usuari) i respostes (respostes predefinides de l'agent que han de començar amb *utter*). Aquest fitxer permet a Rasa saber què pot esperar de l'usuari i què pot respondre l'agent.

config.yml: Aquest fitxer conté la configuració de l'aplicació, que inclou la *pipeline* (la seqüència de pas per al processament del llenguatge natural) i les *policies* (les estratègies que utilitza Rasa per a decidir la pròxima acció a realitzar). La *pipeline* inclou diversos components que permeten extreure intencions i entitats, com ara un *RegexExtractor*, un *WhiteSpaceTokenizer* i el més conegut i emprat, el *DIETClassifier*, mentre que les *policies* poden inclouen estratègies com ara la política de màxim probable, la política de memorització, la política TED, que intenten predir l'acció que ha de prendre l'agent.

[2]YAML: YAML Ain't Markup Language. Són sigles recursives a propòsit.

data/nlu.yml, data/stories.yml, data/rules.yml: Aquests tres fitxers contenen les dades d'entrenament del model. L'nlu.yml conté exemples reals de com els usuaris interactuen amb l'aplicació. Les stories són una representació de la seqüència de diàlegs que es poden donar entre l'agent i l'usuari, mentre que les regles defineixen les seqüències d'accions que el bot seguirà en resposta a certs esdeveniments o estats. Aquests fitxers són crucials per a l'entrenament del model de Rasa.

La idea per desenvolupar un bot conversacional amb Rasa és intentar desplegar l'agent el més aviat millor. Intentar que un model d'aprenentatge automàtic funcioni eficaçment sense moltes dades d'entrenament és molt complicat, per no dir impossible. Per tant, la millor estratègia (i la recomanada des de Rasa) és fer un disseny robust, entrenar-lo amb poques dades en casos pensats per tu, desplegar-lo i que els usuaris el provin. A mesura que el vagin provant, no només obtindràs les seves dades per entrenar el model, sinó que també obtindràs retroacció real de quines funcionalitats estan millor dissenyades i quines són més freqüents.

En el cas d'aquest projecte, estem en el segon pas. Un cop es desplegui la Lovelace es podrà ampliar i entrenar amb dades dels usuaris reals.

Per acabar amb l'estructura d'un projecte de Rasa, parlem ara de l'*Action Server*. Una acció en Rasa és una classe de Python implementant múltiples mètodes. Com a condició, ha d'haver-hi mínim dos d'implementats, el mètode *name* que ha de retornar el nom de l'acció posat al domini i el mètode *run*, que implementa el que l'acció ha de fer en execució.

8.4.2 Connexió Mòduls Xatbot i Rasa

En aquesta secció seran tractat els mètodes pels quals s'ha unit l'arquitectura desenvolupada per l'agent i l'ús de les peticions a l'*Action Server* i al *Tracker* per implementar diverses funcionalitats.

El flux d'un missatge de l'usuari és el següent: Telegram dispara el *webhook* amb els `CustomContext` i `WebhookUpdate` contenint la informació rellevant del missatge. Pel funcionament de l'aplicació, existeixen gestors d'esdeveniments (o *handlers*) que enllacen l'acció que li arriba a una funció (anomenada *callback*) que actua en conseqüència. S'ha delimitat un gestor d'esdeveniments per cada comanda que l'usuari pot emprar (descrites a la secció 7.2.1), per si és un missatge sense comanda, per si és un fitxer amb extensió .c o .java. A la Figura 8.3 es mostra el la seqüència de què succeeix quan s'envia un missatge.

Per tant, el primer pas del sistema és que s'activa una funció o una altra depenent què hagi enviat l'usuari: si és una comanda s'activen les funcions descrites a la secció 7.2.1, si són fitxers amb les extensions predeterminades es realitza la correcció explicada a la secció 7.6 i si és un missatge de text, es crida a la funció `message_handler`. Aquest funció crida a Rasa a l'URL `http://rasa:5005/webhooks/rest/webhook` mitjançant un *webhook*, se'n rep la resposta i s'envia al xat de Telegram.

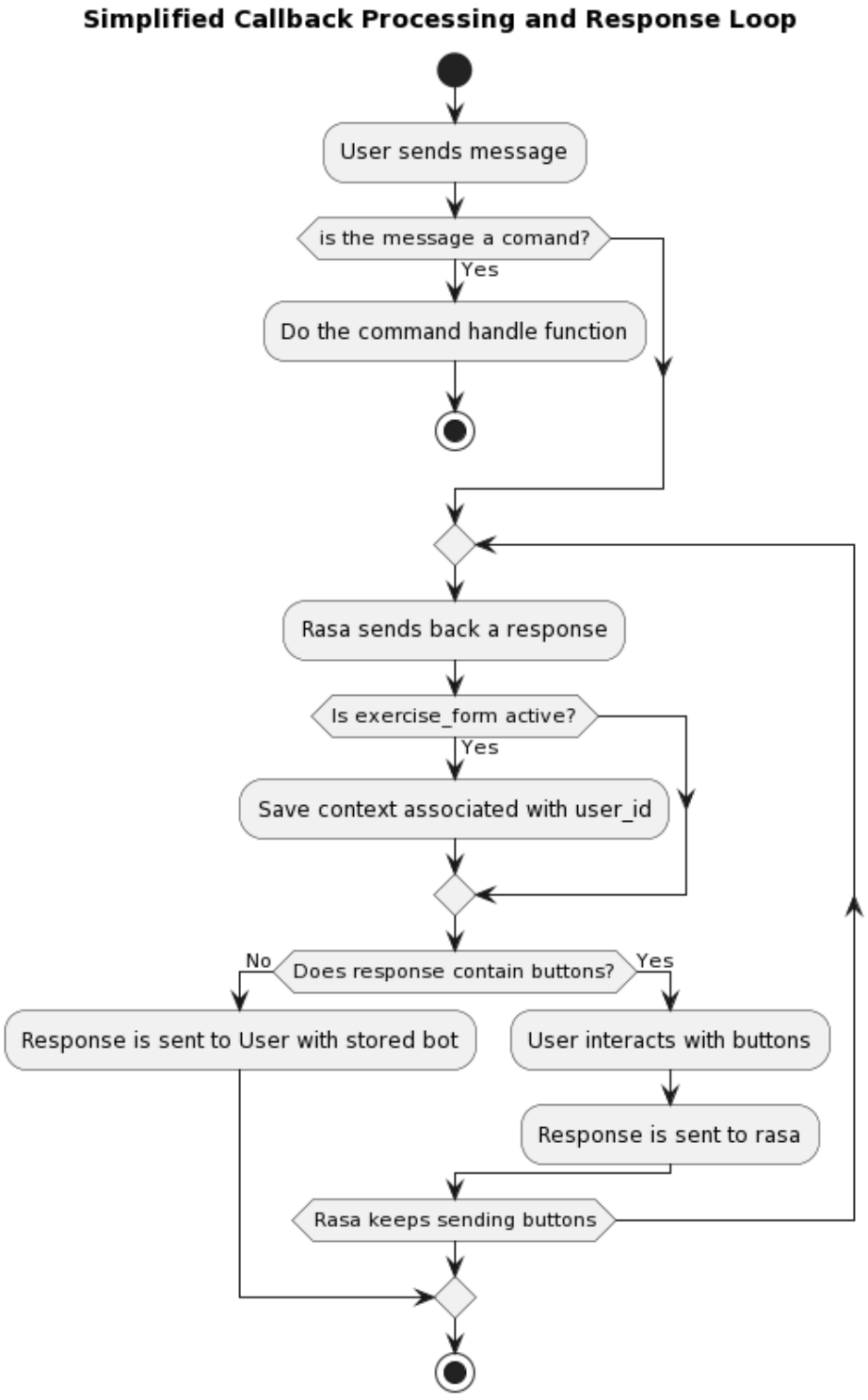


Figura 8.3: Diagrama de seqüència que mostra com es gestiona el missatge d'un usuari, tenint en compte el formulari

A la funció `send_message_to_rasa` però, s'ha de tenir en consideració que l'usuari pot activar el formulari en tot moment, que obligarà a l'agent conversacional a enviar diversos missatges seguits, funcionalitat per la que l'exemple del que es basa aquest projecte no està del tot preparat. Per a aconseguir-ho, s'ha emprat un diccionari per emmagatzemar els contexts just quan s'activa el formulari, recuperar-lo per anar enviant els missatges a l'usuari i eliminar-lo quan s'ha acabat. Aquesta és la casella *is exercise_form active?* de la Figura 8.3.

Quan el model d'NLU ha generat el missatge de resposta, es fa una petició al següent *endpoint* de Rasa: `http://rasa:5005/conversations/user_id/traker`, és a dir, accedim al *traker* de l'usuari concret per comprovar-ne els valors actuals, que venen retornats en una *payload JSON*. En aquesta *payload* no només hi ha una llista dels formularis actius (tenen el bucle activat), sinó que hi ha registrats tots els missatges enviats a la *user_id* concreta, juntament amb les entitats, les ranures, els formularis i molta més informació. En aquest cas, s'extreu de la *payload* si el formulari està actiu o no. El procediment descrit és el mateix que s'ha emprat a la secció 7.3.5 de la Conversa 5: Corregir Exercici però s'extreu el valor de la ranura *subject*.

Seguint amb el flux de treball de la Figura 8.3, si el contingut que envia Rasa no és un missatge de text, sinó que és un botó, es mostren en pantalla i el clic de l'usuari es gestiona amb una altra funció associada amb un *handler*, que envia la *payload* del botó a Rasa. Llavors, es pot generar un bucle en cas que arribin botons de Rasa repetidament. Això significa que estem a dins del context, i quan Rasa de resposta ja no envia més botons, usem el context emmagatzemat per enviar l'últim missatge a la usuària.

8.4.3 Registre de Conversa

Per cada *user_id* de Telegram es genera un fitxer al directori `chatbot/data/logs` que emmagatzema tots els missatges enviats, tant de l'agent conversacional com de l'usuari. Aquesta s'ha implementat mitjançant un *logger*, que es crida amb USR o BOT depenent de qui envia el missatge, tal com es mostra a la Figura 8.4

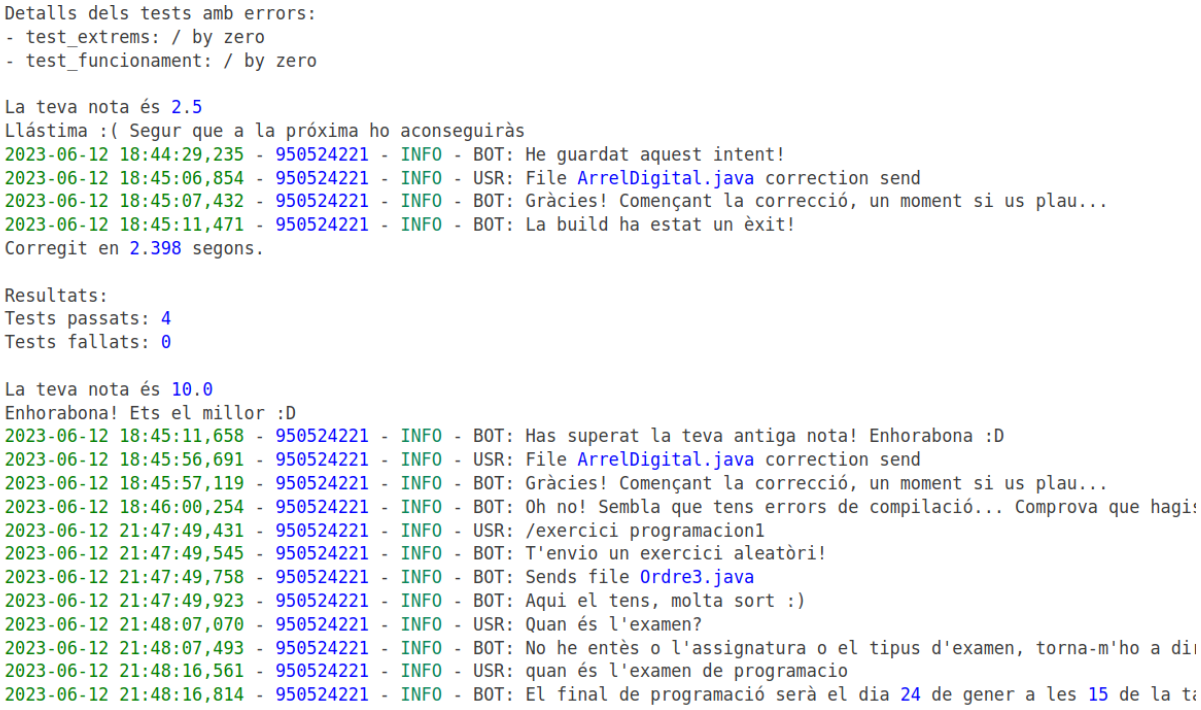


Figura 8.4: Exemple de logs d'un usuari

La implementació darrera aquest fitxer rau en l'inserció del missatge de l'usuari a la `message_handler`, la funció associada al gestor d'esdeveniments (discutit a la secció 8.4.2). A les funcions associades a les comandes de Telegram i a l'enviament del fitxer, s'ha escrit un missatge equivalent a l'ús de la comanda o una frase indicant què ha fet l'usuari. El raonament de no usar el *traker* de Rasa és la implementació per recuperar els fitxers de logs i que no totes les peticions de l'usuari arriben a Rasa.

Aquests fitxers es podran fer servir per a l'entrenament de Rasa amb objectiu de millorar la detecció de les intencions de l'usuari i per analitzar quines són les funcionalitats del bot conversacional més utilitzades. També es podran fer servir per saber en quins punts l'experiència d'usuari no és òptima i quan el bot conversacional té més problemes per entendre l'usuari.

8.5 Actualització de Dades Locals

Aquesta secció tracta dels mecanismes implementats per assegurar-se que les dades de l'aplicació estiguin sempre al dia. Començant per les funcions implementades per poder-ho dur a terme, com funcionen els *webhooks* dels repositoris quan hi ha un *push* i com es mantenen les dades de l'aplicació entre execució i execució.

8.5.1 Clonar Repositoris Locals

A fitxer d'entorn de `GestorDB`, hi ha diverses dades tècniques pel funcionament de l'aplicació, com per exemple els noms dels repositoris que s'han de clonar i al grau al qual pertanyen, tal com s'ensenya a la Figura 8.5:

```
GITHUB_PROGRAMACIO1_REPO=programacion1
GITHUB_REPOS_ASSIGNATURES='[["programacion1", "Informatica"], ["correctorC", "Matematiques"]]'
```

Figura 8.5: Llista d'assignatures utilitzables a l'agent conversacional

Aquest nom ha de coincidir amb el nom del repositori, ja que l'URL de la petició es forma amb `"TOKEN@repositori.git"`, si el repositori és incorrecte l'aplicació no detectarà aquest repositori i seguirà amb la seva execució sense les dades d'aquest.

Pel fet que tenir les dades actualitzades quan l'aplicació engega és crucial per oferir-ne les funcionalitats, s'han implementat una sèrie de redundàncies i reintents automàtics en cas de fallada, tal com es pot veure a la Figura 8.6.

Els passos que segueix el diagrama de la Figura 8.6 són aproximadament els següents.

Primer, comprova si existeix el directori `data/`. Aquest directori és on s'emmagatzemen totes les dades dels repositoris, per tant, s'ha de crear si no hi és. Si `data/` existia - pot haver estat creada per l'usuari o pot haver-se mantingut d'una altra execució- es comproven quins directoris hi ha. Si no hi ha cap directori amb el nom de l'assignatura, es clona el repositori de l'assignatura directament.

Si el directori `data/assignatura/` existeix, s'ha de comprovar que el directori contingui un repositori de GitHub, si no no es podrà fer el *pull*. En cas que ho sigui, es comproven els canvis locals, ja que si n'hi hagués, hi hauria un conflicte amb el *pull*. Si hi ha canvis locals, es fa un git `hard reset` i si no, es fa directament el *pull*.

En cas que el directori `data/assignatura/` no fos un repositori de git, s'elimina el directori es fa un *clone*. Concretament, aquesta funcionalitat s'implementa recursivament, és a dir, el programa elimina el directori i la funció es crida a si mateixa amb el nombre màxim d'intents reduits, on no trobarà cap directori amb el nom que busca i, per tant, clonarà el repositori.

En totes les operacions relacionades amb git explicades als paràgrafs anteriors, hi ha un sistema de reintents. El sistema consisteix en un bucle *while* que es manté actiu augmentant una variable intents i quan intents supera el nombre màxim determinat a la capçalera de la funció s'avança fins a l'estructura de control del final de tot del diagrama.

Per acabar, si la funció ha aconseguit efectuar l'operació amb èxit, imprimeix un missatge d'èxit i s'acaba. En canvi, si per algun motiu no ha pogut executar alguna de les funcions relacionades amb git que s'han mencionat, es crida recursivament amb menys intents. Si

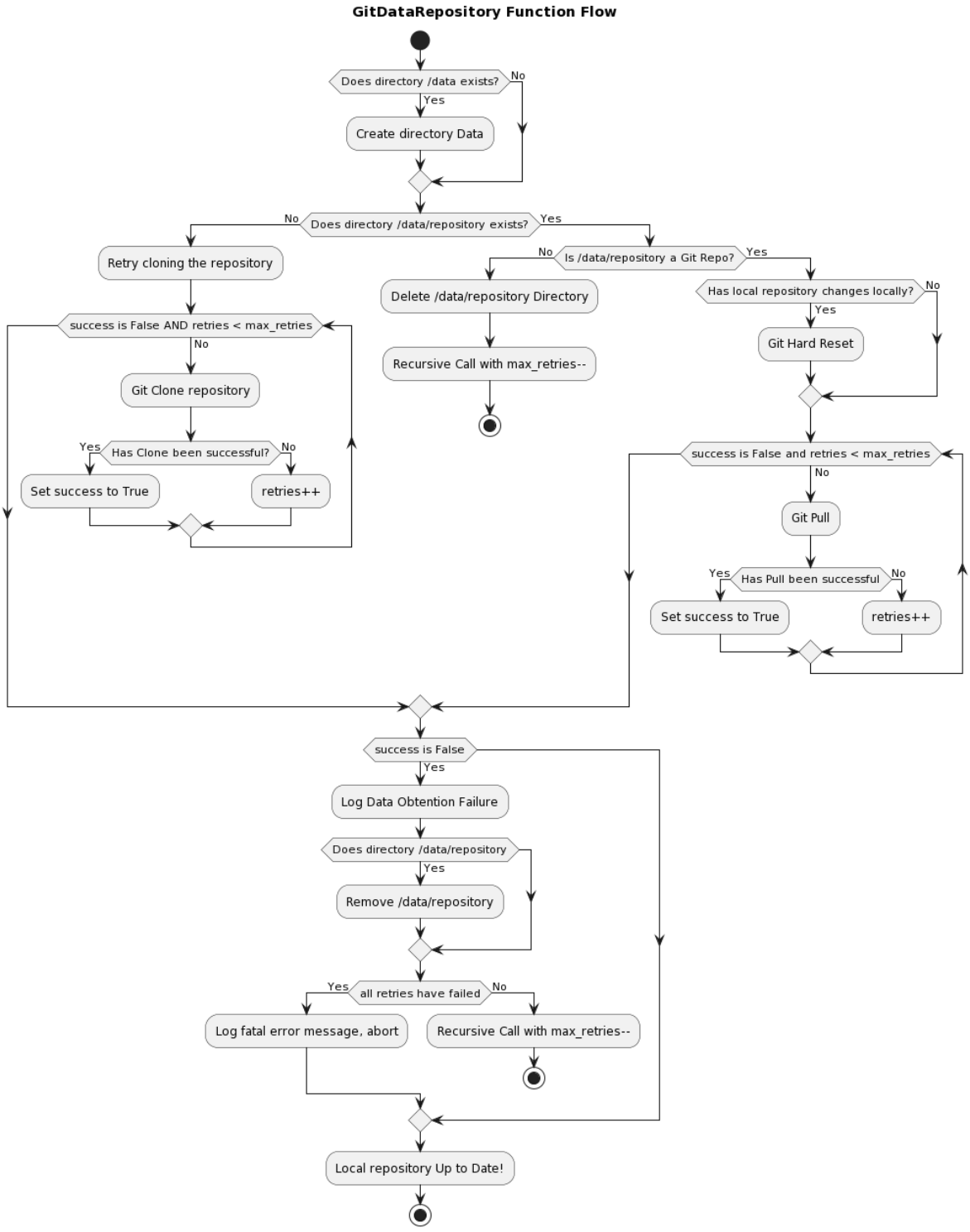


Figura 8.6: Diagrama de la funció Git Data Repo, executada al principi de l'aplicació per aconseguir les dades de l'aplicació

falla en tots els intents, la funció acaba amb estatus de fallada i mata el programa, ja que no podria oferir les funcionalitats adequadament.

Cal mencionar que les referències a `git pull` i `git hard reset` empren les funcions descrites a les seccions 8.5.2 i 8.5.3 implementades amb *gitpy2* mentre que *git clone* i comprovar si un directori és un repositori són de la llibreria *gitpy2*. S'ha decidit implementar les funcions en una llibreria especialitzada de git per Python ja que comporta molts avantatges respecte a emprar la comanda *subprocess*, tal com feien els anteriors projectes. Els avantatges són: 1) una crida a un subprocés és menys eficient que quedar-se en la mateixa execució, 2) el desplegament de l'aplicació pot quedar compromès ja s'ha de garantir que la màquina té accés a un Linux amb git instal·lat, amb la versió i sintaxi correctes respecte a 3) implementar control d'errors via subprocés és més complicat de realitzar des de l'aplicació principal.

8.5.2 Git Pull

Un `git pull` d'un repositori, comprova l'estat del repositori local contra el repositori remot. Si el local està per darrere en canvis, aquests es baixen del repositori remot al local. En altres paraules, actualitza el local si hi ha hagut canvis al remot.

Aquesta funció comprova primer tots els noms de les branques del repositori remot amb el local. Si alguna coincideix amb el repositori clonat, es procedeix a cercar (*fetch*) els canvis al del repositori remot. Un cop els canvis estan localitzats, s'intenta fer un *merge* al repositori local. Llavors la funció pren tres rumbs depenent de quin sigui l'estat del *merge*:

- Estat `UP TO DATE`: si els canvis en local estan *up to date*, surt de la funció.
- Estat `MERGE ANALYSIS FAST-FORWARD`: es comprova si el repositori local es pot dur directament a l'estat del repositori remot sense realitzar cap acció. En cas que es pugui, es busca la branca main del remot, s'actualitza el *commit* de la branca actual i també el *HEAD*, donant per finalitzat el *pull* amb èxit.
- Estat `MERGE ANALYSIS NORMAL`: Aquest cas detecta si no es pot fer un *merge* automàtic, i es necessitaria intervenció d'un humà per resoldre els conflictes entre les dues versions. Aquí entra en joc la variable booleana `force`
  - Si `force` està activada, es crida a la funció de gitpy2 *merge*, que posa els canvis del remot en local.
  - Si `force` no està activada, s'acaba l'execució del programa i el *merge* no està actualitzat, llençant un *AssertionError*.

La decisió d'incloure el paràmetre force per, en cas de conflictes sobreescriure els canvis en local és per seguretat; la màquina que hosteja el projecte no ha d'aportar mai canvis als repositoris de manera automàtica, així que sempre els donen prioritat als canvis externs.

8.5.3 Git Hard Reset

La funció `git_hard_reset` s'ha implementat com a mesura de seguretat. Podria passar que algú canviés un repositori local (afegís un fitxer, canvies alguna línia) sense voler a la màquina host. Al llavors comprovar si s'han d'actualitzar els repositoris a l'inici del programa (vegis la següent secció per context) no es podrien dur a terme sense fer un hard reset prèviament o sense forcar el git pull, ja que donaria error si hi haguessin canvis no contemplats.

Aquesta funció torna el repositori local a l'estat del repositori remot, sense importar quins canvis hi hagi al local.

La funció `hard_reset` és molt senzilla en comparació a l'anterior. Com en el *pull*, primer s'obté la branca *origin*, que és la predeterminada per defecte quan clones un repositori. Llavors es cerca (*fetch*) els canvis al repositori i es comproven els dos últims *commits*, el del clon local i el de GitHub. En el cas que coincideixin, se surt de la funció, i si no, es crida la funció `reset` de gitpy2 amb *GIT_RESET_HARD* i la id del *commit*, que fa tornar el repositori local a l'últim *commit* registrat a GitHub.

8.5.4 Notificacions *Push* via *Webhook*

L'aplicació es manté al dia perquè a tots els repositoris de Github contenint el codi dels exercicis, s'ha creat un *webhook* apuntant a l'*endpoint* "`/hook`"de `GestorDB`, que es dispara cada vegada que un dels repositoris rep un *push*, tal com es veu en la Figura 8.7.

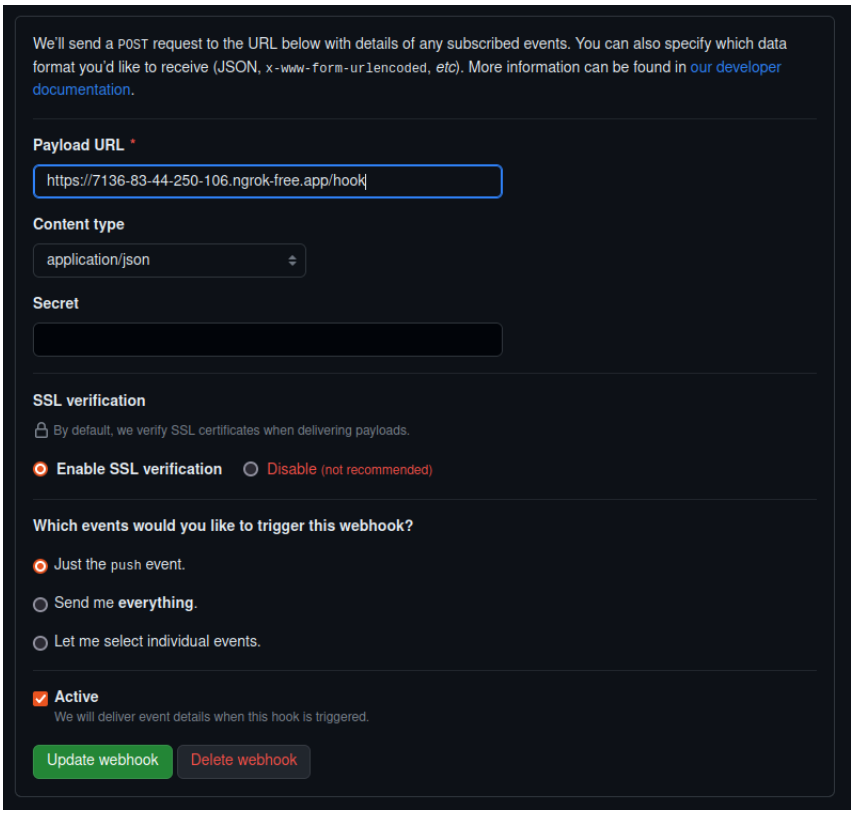


Figura 8.7: Com posar el webhook de github

Llavors, si a `GestorDB` li arriba un *webhook*, processa la *payload* ajuntant tots els fitxers modificats, afegits i eliminats de tots els *commits* fins a quedar-se amb tots els canvis a realitzar, i en canvia la base de dades local en conseqüència.

Per processar correctament les *payloads* dels *push*, s'han creat tres objectes heretant de *WebhookCommonPayload* que són els següents:

- Classe `Commit`: conté la id de l'usuari que ha fet, el *commit*, l'URL, el nom d'usuari i llistes dels fitxers afegits, modificats i esborrats per *commit*.
- Classe `Repositori`: conté la id del repositori, juntament amb el nom, el propietari i el nom sencer d'aquest.
- Classe `PushPayload`: Conté les dues classes anteriors més informació sobre general del *pull*, com la llista de *commits*, el *commit* principal i quin usuari ha fet el *push*.

Gràcies a aquestes estructures niuades, l'actualització de la base de dades al rebre un *Push* itera sobre els *commits*, destriant tots els fitxers modificats, afegits i eliminats i actualitzant-los a la base de dades. En consideració de la claredat, GitHub considera

modificació a un fitxer exclusivament canvis en el seu interior, mentre que canviar de nom o de directori és eliminar l'antic i afegir el nou. Això simplifica molt actualitzar-la en un *push*, ja que només s'han d'eliminar tots els fitxers de la llista i afegir els altres.

8.5.5 Persistència de Dades entre Execucions

La connexió a MariaDB a través de `GestorDB` recau en *DatabaseManager*, que comprova si la base de dades existeix i connectar-s'hi o crear-la si no existeix. En cas de voler que les dades de la base de dades no es conservin entre execucions (per testatge o caigudes del sistema un cop ja desplegat) s'ha implementat un paràmetre a l'entorn arrel de l'aplicació anomenat `RESET_DATABASE`. Si aquest és True, trunca totes les taules de la base de dades i les deixa buides, i si és False, no elimina les dades relacionades amb els usuaris.

Pel disseny del sistema, si `RESET_DATABASE` és False també trunca les taules de *exercise*, *test* i *subject* i quan estan els repositoris clonats torna a omplir la base de dades amb ítems que ja conté, donant error. La manera de gestionar aquest procés seria no descarregar les dades a l'inici per defecte i que comparar l'estat de la base de dades amb el repositori local, no repetir el curs normal del programa.

Malauradament, el procediment descrit al paràgraf anterior no s'ha realitzat, ja que canviar la lògica d'aquest codi no és senzill i pel marc temporal inherent a aquest projecte, s'ha decidit destinar els recursos a altres funcionalitats. Concretament, es podrien reaprofitar les funcions d'actualització de la base de dades, que ja estan fetes per escoltar *webhooks* en *push*, però s'hauria d'implementar una altra funció que retornes les diferències entre el repositori clonat en local i la base de dades per poder-los passar a la funció d'actualitzar.

8.6 Privacitat

En aquesta secció s'explica les consideracions preses pel tractament de dades delicades dels usuaris. Concretament, s'ha implementat un algorisme *hash* SHA265 per les *telegram_user_id* i un sistema d'encriptació *AES*. Les funcions *hash* són funcions que transformen un element en un altre (normalment de longitud fixa) de manera no bijectiva, és a dir, que no hi ha una correspondència un a un amb l'element original. L'*AES* (Advanced Encryption System) en canvi, és un algorisme de xifratge simètric mitjançant claus, és a dir, es pot desfer per obtenir les dades d'entrada si es té la clau que s'ha usat per encriptar-les.

8.6.1 SHA256 amb *Salt*

Per implementar el *Hashing*, hem usat la funció *SHA256* [3] de la llibreria `hashlib` de Python. L'hem implementat amb l'anomenada *salt*, que consisteix a afegir una cadena de 16 bits al final de cada *user_id* abans de passar-lo per la funció, vegis el perquè a la secció 8.7.1

La implementació del *hashing* en l'emmagatzematge del *user_id* a la base de dades comporta un augment de complexitat tant en cercar un usuari. En primer lloc, òbviament hem de *hashejar* els *user_id* que volguem cercar. Així i tot, cercar la cadena a la base de dades directament no funcionaria, ja que la *salt* canvia, fent que la imatge de la *SHA256* sigui diferent per un mateix id.

L'estratègia apropiada per a aquest escenari consisteix en *hashejar* el *user_id* amb les *salts* emmagatzemades de cada usuari prèviament a memòria. Si el resultat del *hash* amb l'antiga *salt* d'algun dels usuaris coincideix amb algun dels *hashed_id* emmagatzemats a la base de dades, aleshores hem identificat un usuari existent. Òbviament, això implica la necessitat de guardar també les *salts* de tots els usuaris, incrementant així l'espai de memòria requerit per a la nostra implementació.

L'aplicació segueix l'aproximació narrada als paràgrafs anteriors; concretament, fa els següents passos:

1. L'usuari entra la comanda `/entra nom correu`
2. `Xatbot` reencamina a la funció *entra()*. Aquesta fa una petició *check_user(user_id)* a `GestorDB` per saber si l'usuari es troba a la base de dades.
3. `GestorDB` crida a *verify_user_id()* la qual fa el següent:
    - Recupera tots els *hashed_id* i les *salts* de tots els usuaris emmagatzemats a *user_security*
    - Hasheja el *user_id* del nou usuari amb la *salt* emmagatzemada a la base de dades. Si aquest coincideix amb el *hashed_id* corresponent a la salt, s'ha trobat l'usuari a la base de dades, si no n'hi ha cap que coincideixi, l'usuari no havia entrat abans.
4. `GestorDB` envia els resultats a `Xatbot`, que continua amb l'execució prevista.

Un cop l'usuari ha estat guardat a la base de dades, se li assigna una id numèrica autoincremental per facilitar la cerca i la gestió d'usuaris en l'àmbit intern, perquè tot i que identificar-los amb la *hashed_id* és possible i proporciona una identificació inequívoca, és molt feixuga de tractar, ja que té 256 bits de longitud.

[3] SHA256: Secure Hash Algorithm 256-bits. Els bits del nom indiquen els bits de sortida.

8.6.2 Advanced Encription System (AES)

L'AES és un estàndard de xifratge simètric[4] amb clau. En aquest projecte emprem una clau de 128 bits. Pel correcte funcionament de l'algorisme es necessiten els paràmetres IV (inici de vector) i *tag* (etiqueta). L'inici de vector és una seqüència binària pseudoaleatòria que s'usa per garantir que el resultat és únic per cada entrada, tot i que aquesta sigui la mateixa. La *tag* són dades generades durant el xifratge que s'usen per comprovar la integritat de l'entrada, és a dir, és un paràmetre de control. L'IV va associat a la clau que s'usa, mentre que la *tag* depèn de cada conjunt de dades a encriptar.

En el context del projecte, els correus s'encripten amb AES per a ser desencriptats i fets servir per futures funcionalitats, com per exemple, enviar els exercicis a alumnes per correu, vegis secció 10 per a més. En l'estat actual no s'usa el correu de l'usuari en cap funcionalitat ja que no s'han pogut implementar per falta de temps. Les funcions d'encriptació i desencriptació estan implementades amb funcions de la llibreria `cryptography` de Python i han estat testejades tot i el seu poc ús.

8.6.3 Tractament de les Dades

S'ha intentat minimitzar el recorregut de les dades no encriptades internament a l'aplicació, és a dir, seguint una política d'encriptar el més aviat millor. Quan un correu i una id s'afegeixen a base de dades, `Xatbot` les envia encriptades ja directament a través de l'*endpoint* a `GestorDB`. Concretament, s'envien la id *hashejada*, la *salt* associada a la id, el correu encriptat, l'IV i el *tag*. Tots els paràmetres d'AES són convertits a cadena de text abans de ser enviats per l'*endpoint* per no tenir problemes de comunicació inesperats, però són reconvertits a bytes en el moment del seu emmagatzematge.

En cas de consultar si un usuari és a la base de dades o buscar la seva id interna associada, no podem *hashejar* la *user_id* en cap moment, si no, no es podria trobar-lo.

8.7 Seguretat

En aquest projecte s'ha tingut en compte la seguretat tant de les dades de l'usuari com les potencials bretxes en diverses implementacions. En aquest apartat justifiquem les decisions i donem detalls rellevants d'aquestes.

[4]Simètric significa que amb la clau privada és desencriptable

8.7.1 Encriptació i Hashing

Com ja s'ha vist a la secció 8.6, dos algoritmes de xifrat han estat implementats, SHA256 amb *salt* i de AES amb clau de 128 bits. En aquest apartat s'inclouen les justificacions de seguretat de per què s'han pres aquestes decisions.

La implementació de *hashing* per la identificació d'usuaris és més que comuna i àmpliament usada en una gran diversitat de projectes, sent una gran pràctica per garantir l'anonimat dels usuaris. Addicionalment, hem emprat la *salt* per fer el sistema més resistent a l'anomenat *Rainbow Attack*. Els *Rainbow Atacks* consisteixen a provar moltes entrades i sortides de taules hash públiques a internet per veure si trobant prou coincidències, es pot endevinar la funció *hash* implementada. Evidentment, si un agent extern troba la funció *hash* emprada pel *hashing* podria fer-se passar per altres usuaris o comprometre informacions delicades.

La decisió de hashejar els *user_id* està motivada per la necessitat de protegir l'anonimat dels usuaris, donat que la identificació d'usuari de Telegram pot ser utilitzada per traçar l'activitat d'un usuari en aquesta plataforma. Aplicar el *hashing* fa que trobar i identificar usuaris a la base de dades sigui una tasca lleugerament més enrevessada tan computacionalment com d'implementació.

En el cas dels correus electrònics és també àmpliament sabut que les dades delicades dels usuaris s'han d'emmagatzemar encriptades en cas de qualsevol bretxa de seguretat. S'ha escollit AES, ja que aquest sistema d'encriptació és àmpliament reconegut pel seu equilibri entre seguretat i eficiència, i és ideal per a la protecció de dades que han de ser emmagatzemades a llarg termini, com és el cas dels correus electrònics.

8.7.2 Avantatges de l'Execució amb Docker

Executar codi de terces sense comprovacions prèvies o supervisió no és sota cap cas segur: En primer lloc, aquest codi pot contenir intencions malicioses, obrint una potencial bretxa de seguretat i l'execució d'aquest codi en la màquina local podria comprometre el sistema i exposar informació delicada. En segon lloc, tot i el codi no ser maliciós *perse*, pot malmetre el sistema si s'executen programes no previstos pel programador, cosa que també podria posar en risc l'execució i la seguretat del sistema.

En aquest context, l'ús de Docker es presenta com una solució molt adequada. Docker permet una execució completament aïllada de la màquina local, fet que redueix dràsticament el risc associat a l'execució de codi potencialment maliciós. Aquesta execució aïllada redueix significativament les possibilitats que qualsevol atac a través del codi executat pugui afectar el sistema local.

Es destaca que l'accés de Docker a la màquina local és inexistentment o extremadament limitat, especialment si es controlen els permisos d'escriptura al Docker. Aquesta carac-

terística addicionalment contribueix a la seguretat del sistema, ja que restringeix qualsevol possible efecte inesperat de l'execució del codi.

En el context del nostre projecte, no s'han donat permisos d'escriptura als processos generats amb *Dockerpy*, així que ni que un agent maliciós s'aprofités dels correctors per entrar al projecte, hauria d'accedir a la màquina local amb només memòria, sense poger emmagatzemar res.

Per tant, l'elecció de Docker per a la correcció de les implementacions dels estudiants no és trivial, sinó una decisió calculada per minimitzar riscos i garantir la seguretat del sistema enfront de possibles codis maliciosos o comportaments inesperats que puguin causar un error fatal [5] en l'execució.

8.7.3 Seguretat de Dockers Aniuats

Cal justificar l'elecció de per què només s'han *dockeritzat* quatre dels cinc servidors totals de l'aplicació (`Xatbot`, `GestorDB`, `Rasa`, `Action Server` i `MariaDB`) si *docker-compose* permet gestionar a la vegada tantes imatges com el dissenyador precisi. La raó principal d'aquesta decisió és que el mòdul `Corrector`, a diferència dels altres dos, ha de crear i gestionar contenidors Docker per si mateix, ja que proporciona l'execució de les correccions.

Un contenidor empra els recursos del sistema operatiu per acomplir les seves tasques, això és el que li permet ser molt més lleuger que una màquina virtual i igual de versàtil en aïllament de l'execució i dependències. Pel contenidor poder fer peticions al nucli del sistema operatiu ho fa a través de *sockets*[6], concretament, cada contenidor té un *socket* que li permet comunicar-se amb el sistema operatiu. Quan creem un contenidor dins d'un altre, s'ha d'aconseguir que el *socket* del contenidor dins del contenidor que l'ha creat sigui accessible pel sistema operatiu.

Tot i que donar accés al *socket* del contenidor intern és tècnicament possible compartint el *socket* del nou contenidor amb el sistema operatiu mitjançant un volum, comporta riscos significatius de seguretat que no poden ser ignorats. En particular, compartir el *socket* del contenidor intern dóna al contenidor intern un control total sobre el contenidor extern. Això significa que si el contenidor intern es veiés compromès, l'atacant podria prendre el control total de la màquina local, una conseqüència amb un risc de seguretat extremadament alt.

Cal recordar que l'essència de l'ús de contenidors està en la seva capacitat d'aïllament: cada contenidor hauria de funcionar com una unitat independent i autònoma. La capacitat d'un contenidor per crear i administrar altres contenidors pot erosionar aquesta barrera

[5] Error Fatal: error irrecuperable en execució, però que no malmet el sistema.

[6] Socket: concepte abstracte mitjançant el qual dos programes poden fluxos de dades de manera fiable i ordenada.

d'aïllament. A més, cal esmentar que la tecnologia Docker original no va ser dissenyada amb aquesta funcionalitat en ment, i per aquesta raó, quan un contenidor Docker s'encarrega de la creació i gestió d'altres contenidors, es poden produir riscos de seguretat significatius que no es van preveure en el disseny inicial d'aquesta tecnologia.

Per tant, per respectar els principis de seguretat i aïllament i evitar possibles atacs que podrien comprometre tot el sistema, s'ha optat per no *dockeritzar* el mòdul `Corrector`. Aquesta decisió augmenta la seguretat general de l'aplicació i assegura una millor adhesió als principis de la tecnologia de contenidors.

Capítol 9

Conclusions

Aquest projecte ha aconseguit presentar la LovelaceUB, una agent conversacional de suport a l'aprenentatge multiassignatura amb capacitats de comprensió del llenguatge natural, generació de llenguatge natural mitjançant *GPT3.5-turbo*, i execució de codi Java i C per la correcció dels exercicis enviats pels estudiants. També s'ha assolit formalitzar, aprofundir, implementar i dissenyar tests sobre mètodes numèrics ensenyats al grau de matemàtiques, juntament amb una llibreria per facilitar la inclusió dels futurs enunciats, exercicis i tests per part dels docents del grau.

Igualment s'ha assolit donar una bona experiència d'usuari mitjançant l'aprenentatge automàtic usant la tecnologia Rasa en la detecció d'intencions de l'usuari, produint-se aquesta interacció en una interfície coneguda i popular com Telegram, a part de facilitar l'actualització de continguts al sistema mitjançant *webhooks on push* als repositoris de les assignatures.

Malauradament, per assolir aquest projecte, s'ha hagut de descartar tot el treball previ dels altres dos projectes [5] [6] a causa de problemes en el codi original i la naturalesa de l'abast del nou projecte han resultat ser molt diferents, descartant la reutilització de material. També, un dels objectius lligats a la implementació d'un marc conversacional amb funcionalitats d'aprenentatge automàtic era el canvi de disseny de com s'interactuava amb l'agent original, reforçant encara més la tesi de començar el projecte de zero. Les úniques coses que s'han reaprofitat han estat les idees de disseny de com demanar exercicis i l'ús de contenidors per garantir una execució segura.

No s'ha pogut assolir el desplegament de l'aplicació en un servei de *hosting*, tot i l'aplicació estar perfectament preparada perquè es faci. Els motius radiquen en el marc temporal inherent a un projecte d'aquestes característiques i, sobretot, l'augment inesperat de la complexitat d'aquesta aplicació. Fer un desplegament de sis servidors, un d'ells generant contenidors de Docker a voluntat, no és una empresa trivial, i menys per fer un desplegament de producció, el qual implicaria especificar la gestió de recursos al servei

d'allotjament que es decidís emprar.

Tot i tenir una gran quantitat de treball futur, tal com s'explica a la següent secció, es creu que els objectius d'aquest projecte, tot i la seva extensió i dificultats imprevistes, dona una sòlida base per seguir-ne ampliant el contingut i desplegar-lo el més aviat millor pel seu testatge amb usuaries reals.

Capítol 10

Treball Futur

El projecte de la LovelaceUB ha crescut inexorablement d'ençà que va començar, deixant molts fronts oberts per la millora i l'ampliació.

CUnit: Tal com ja s'ha explicat en profunditat a la secció 7.7.3, CUnit no és la llibreria que millor s'adapta a les necessitats d'aquest projecte. Una millora seria traduir la llibreria descrita a la secció 7.7 per la correcció de problemes matemàtics per a que fes ús de les funcions de Criterion[17], tanmateix com canviar el contenidor que crea *Corrector* per recollir els resultats adequadament.

Funcionalitats anteriors: Els dos anteriors projectes [6] [5] implementaven diverses funcionalitats que no s'han pogut traslladar a aquest projecte per la gestió de prioritats i la falta de temps. Entre altres, una funcionalitat interessant és enviar els exercicis demanats per les estudiants a als seus correus en comptes de d'enviar-los a través del xat. També, seguint la mateixa línia de raonament, es podria enviar certa informació per correu, com els plans docents o respostes generades a dubtes concrets de l'estudiant.

GestorDB i la Informació: Es podria desenvolupar un sistema perquè l'aplicació tingui accés a les dades d'exàmens i pla docent de l'assignatura. Aquest sistema podria ser un repositori anomenat *InformacióAssignatures* on es continguin les dates, criteris d'avaluació i altres dades de les assignatures que inclou el bot, o alternativament, fer peticions al servidor de la UB (cosa que no depèn d'aquest projecte). Una altra acció relacionada amb el `GestorDB` seria seria implementar taules noves a la base de dades (o altres mecanismes de persistència de dades) que s'omplissin des del mateix repositori de l'assignatura per obtenir els valors dels botons que es mostren als usuaris. Com per exemple, el tipus d'exercici varia depenent de l'assignatura (Iteratives i SequencialsAlternatives de Programació I contra InterpolacióPolinomial i MètodePotència de Mètodes Numèrics) i que mitjançant aquesta funció canvies el que apareix als botons, que per les limitacions temporals inherents a aquest projecte, això s'ha deixat estàtic al codi, no amb peticions.

Autocompletat Telegram: Tal com s'ha mencionat a la secció 7.3.7, afegir un sistema

d'autocompletat seria molt bo per millorar l'experiència d'usuari amb les comandes. Si no, emprar la tecnologia de Rasa per buscar sinònims al fer les peticions (per exemple, que l'agent comprengui que programació, programació 1 i programació I es refereix a la mateixa entitat).

GPT3: Aquí és on, indubtablement, rau el potencial d'aquesta agent conversacional. En un afany de mostrar el concepte, només s'ha implementat un sistema de dubtes relativament senzill, però molt ampliable a moltes altres capacitats de l'agent. Per exemple, en comptes de tenir frases predeterminades per saber on o quan és l'examen d'una assignatura, es podria només detectar les entitats, fer petició a `GestorDB` per aconseguir les dades i fer una petició a GPT demanant construir una frase que uneixi el termes donats. Més enllà d'això, també es podria fer servir com a *fallback* en cas que Rasa no entengués la petició de l'usuari, passant-li a GPT i ell escollint l'acció que creu més apropiada. Per acabar, i fer l'experiència d'usuari més completa, se li podria donar accés als últims missatges que han intercanviat l'agent i l'usuari, per comprendre millor la petició d'aquest.

Disseny del Bot: Per acabar, tot i aquest projecte donar una implementació que assoleix ser multiassignatura, no és l'èmfasi d'aquest projecte dissenyar el bot final, sinó donar-ne una base i idea. Per tant, el treball futur per la LovelaceUB és dissenyar com es tracten les assignatures exactament, si hauria d'haver-hi diferents bots, un per cada assignatura o que tot estigués contingut en un amb indicadors clars per l'usuari de quina assignatura està.

Bibliografia

[1] Alfio Quarteroni, Riccardo Sacco, F. S. *Numerical Mathematics*. Springer, New York, 2000.

[2] Atkinson, K. E. *An Introduction to Numerical Analysis*, second ed. John Wiley & Sons, New-York, Chicester, Brisbane, Toronto, 1989.

[3] Docker. `https://docs.docker.com/get-started/overview/`. Accedit: 31-05-2023.

[4] Docker. `https://docs.docker.com/storage/volumes/`. Accedit: 31-05-2023.

[5] Domenech Puig, J. Dockers para un sistema de corrección de ejercicios en un chatbot. *Universitat de Barcelona* (2022).

[6] Fernández Fernández, D. Agentes conversacionales educativos. *Universitat de Barcelona* (2021).

[7] github-webhook framework. example.py. `https://pypi.org/project/github-webhooks-framework/`. Accedit: 22-02-2023.

[8] hookdeck.com. Introduction to asynchronous processing. `https://hookdeck.com/blog/post/introduction-asynchronous-processing#conclusion`. Accedit: 11-06-2023.

[9] hookdeck.com. Why implement asynchronous processing of webhooks. `https://hookdeck.com/webhooks/guides/why-implement-asynchronous-processing-webhooks#when-ingesting`. Accedit: 11-06-2023.

[10] hookdeck.com. Why you should stop processing your webhooks synchronously. `https://hookdeck.com/webhooks/guides/why-you-should-stop-processing-your-webhooks-synchronously#webhooks-are-events`. Accedit: 11-06-2023.

[11] Khosrawi-Rad, B., Rinn, H., Schlimbach, R., Gebbing, P., Yang, X., Lattemann, C., Markgraf, D., and Robra-Bissantz, S. Conversational agents in education – a systematic literature review.

[12] Lars Eldén, Linke Wittmeyer-Koch, H. B. N. *Introduction to Numerical Computation*. Studentlitteratur AB, June 2004.

[13] Mantha, M. https://rasa.com/blog/introducing-dual-intent-and-entity-\transformer-diet-state-of-the-art-performance-on-a-lightweight\-architecture/. Accedit: 31-05-2023.

[14] python-telegram bot. customwebhookbot.py. https://github.com/python-telegram-bot/python-telegram-bot/blob/master/examples/customwebhookbot.py. Accedit: 22-02-2023.

[15] Rasa. customwebhookbot.py. https://rasa.community/open-source-nlu-nlp/. Accedit: 22-02-2023.

[16] Richard L. Burden, J. D. F. *Numerical Analisis*, ninth ed. Brooks/Cole, Cengage Learning, United States, 1989.

[17] Snaipe. Criterion. https://github.com/Snaipe/Criterion. Accedit: 12-06-2023.

[18] Source, R. O. https://rasa.com/docs/rasa/architecture/. Accedit: 31-05-2023.

[19] Source, R. O. https://rasa.com/docs/rasa/policies/. Accedit: 31-05-2023.

[20] TERMCAT. https://www.termcat.cat/ca/cercaterm/fitxa/MzU2ODQ1Mg%3D%3D. Accedit: 08-06-2023.

[21] UniversitatBarcelona. https://www.ub.edu/sens-dubte/consultes/xatbot/. Accedit: 08-06-2023.

[22] UniversitatObertaCatalunya. https://www.uoc.edu/portal/ca/servei-linguistic/criteris/lexic/tractament-anglicismes/index.html. Accedit: 08-06-2023.

[23] Viquipedia. https://ca.wikipedia.org/wiki/ELIZA. Accedit: 04-03-2023.

[24] Viquipedia. https://ca.wikipedia.org/wiki/Model_de_llenguatge_gran. Accedit: 04-03-2023.

[25] Viquipèdia. https://ca.wikipedia.org/wiki/Dialogflow. Accedit: 08-06-2023.

[26] Viquipèdia. https://ca.wikipedia.org/wiki/Watson_(intel%C2%B7lig%C3%A8ncia_artificial). Accedit: 08-06-2023.

[27] Viquièdia. https://ca.wikipedia.org/wiki/OpenAI. Accedit: 13-06-2023.

[28] Wikipedia. https://en.wikipedia.org/wiki/Software_design_pattern. Accedit: 15-05-2023.

[29] Wikipedia. Solid principles. https://en.wikipedia.org/wiki/SOLID. Accedit 15-05-2023.

Apèndix A

Manual Tècnic

A.1 Requisits de Software

El desenvolupament del projecte s'ha fet en Ubuntu 22.04.2 LTS. Les versions de tots el programaris necessaris per l'execució del projecte *dockeritzat* són les següents:

- Python 3.10.6
- docker 20.10.21
- docker-compose 1.29.2

En canvi, si es vol correr sense emprar Docker-Compose (en local), sorgiran, com a mínim, les següents dependències:

- MariaDB 10.6.12
- ConnectorC/MariaDB (última versió)
- rasa 3.5.8

Es recomana emprar docker-compose per l'execució del codi i no barallar-se amb les dependències en local si no s'ha de modificar el codi.

Totes les llibreries de Python emprades en cada mòdul tenen el seu `requirements.txt`, on consten altres mòduls mencionats a tecnologies (secció 8.1) com FastAPI i gunicorn amb les seves versions corresponents. És altament recomanable que cada mòdul s'instal·li en el seu propi entorn virtual amb la comanda `pip install -r requirements.txt`

A.2 Estructura

L'arrel del projecte conté 4 carpetes: `/chatbot`, `/manager_db`, `/corrector`, i `/rasa`. Les dues primeres contenen els mòduls descrits a la memòria com `Xatbot` i `GestorDB`, la segona conté el sistemes de correció (el `Corrector` de la memòria), els Dockerfiles i projectes dels correctors de C i de Java i `rasa` conté el bot de rasa (NLU) i el *rasa action server* dins de `rasa/actions`

Els directoris `/chatbot`, `/manager_db`, `/rasa`, `/corrector/dockers/maven`, `/corrector/dockers/cunit` contenen Dockerfiles per la construcció dels contenidors. Aquests, si són canviat de directori, deixaran de funcionar sense fer-los-hi els canvis requerits.

Estructuralment el projecte funciona amb quatre entorns virtuals de Python, un per cada directori arrel mencionat al primer paràgraf d'aquesta secció. Al directori de rasa l'entorn s'usa per executar tan el servidor d'*NLU* com l'*Action Server*, fent un total de cinc processos actius en local per fer anar tot el projecte.

Tot i tenir cada mòdul un entorn associat, i per tant tenir el corresponent fitxer de variables d'entorn `.env`, n'hi ha un a l'arrel de projecte, juntament amb el *docker-compose.yml*, que s'usa per carregar totes les variables d'entorn compartides a tot el projecte. La descripció d'aquestes es troba a continuació:

```
BUILD=local
#BUILD=docker
#BUILD=prod

CHATBOT_LOCAL_HOST = 0.0.0.0
CHATBOT_DOCKER_HOST = chatbot
CHATBOT_PORT = 8443

MANAGERDB_LOCAL_HOST = 0.0.0.0
MANAGERDB_DOCKER_HOST = manager_db
MANAGERDB_PORT = 5001

CORRECTOR_LOCAL_HOST = 0.0.0.0
IP_LOCAL_MACHINE = 172.20.0.1
CORRECTOR_PORT = 5002

RASA_LOCAL_HOST = 0.0.0.0
RASA_DOCKER_HOST = rasa
RASA_PORT = 5005

ACTION_SERVER_LOCAL_HOST = 0.0.0.0
ACTION_SERVER_DOCKER_HOST = action_server
ACTION_SERVER_PORT = 5055

MARIADB_PORT = 3306

```

```
MYSQL_DATABASE=chatbotDB
MYSQL_USER=manager
MYSQL_PASSWORD=#A escollir
MYSQL_DOCKER_HOST=db
MYSQL_LOCAL_HOST=127.0.0.1
MYSQL_ROOT_PASSWORD=#A escollir
RESET_DATABASE = True #o False, especificament majuscula
```

Listing A.1: Variables d'entorn comunes al projecte `/lovelaceUB/.env`

La variable `BUILD` es refereix a l'estat en que es vol executar l'aplicació, i segons aquesta, canviaran els valors de les variables segons continguin `LOCAL` o `DOCKER` al nom.

L'elecció de totes les `LOCAL` a `0.0.0.0` no és trivial, ja que no pot ser *127.0.0.1* (si més no a corrector), ja que sinó no podrà comunicar-se amb els contenidors de correcció dels llenguatges. La `IP_LOCAL_MACHINE` s'ha d'obtenir mitjançant `ifconfig` i és la que s'usa per la comunicació del contenidors de Docker amb el mòdul `Corrector`, ja que la comunicació interna es realitza mitjançant el nom del host. Totes les variables de forma `modul_DOCKER_HOST` han de tenir el valor coincident amb el nom de la imatge definit a `docker-compose.ymal`

A la primera execució de MariaDB s'ha de crear un usuari no *root* i definir-lo aquí, juntament amb la contrasenya de *root* i la de l'usuari.

Es descriu a continuació l'entorn de *chatbot*:

```
TELEGRAM_WEBHOOK=https://481b-85-49-232-201.ngrok-free.app
TELEGRAM_BOT_TOKEN=redacted
ADMIN_CHAT_ID=123456
PRIVATE_KEY=#AES: string aleatori de 256 bits
```

Listing A.2: Variables d'entorn de chatbot `/lovelaceUB/chatbot/.env`

Per apuntar el *webhook* de Telegram a la direcció correcta del nostre servidor, s'ha d'emprar *ngrok*, vegis secció A.3 de com i perquè. Tot i això, està tot descrit al `README` del projecte.

El `TELEGRAM_BOT_TOKEN` es necessari per la connexió del *webhook*, concretament l'URL `https://api.telegram.org/bot<TELEGRAM_BOT_TOKEN>/\setWebhook=<TELEGRAM_WEBHOOK>` és la que s'ha d'emprar (i el codi ho fa així internament) i per comprovar-ne l'estat es pot fer amb la següent URL: `https://api.telegram.org/bot<TELEGRAM_BOT_TOKEN>/getWebhookInfo`. No es proporciona el valor de `TELEGRAM_BOT_TOKEN` ja que és informació delicada, és la única manera que un potencial agent maliciós fes peticions al sistema. Si es vol executar el codi amb connexió en Telegram, contactis amb els autors.

Es descriu a continuació la variable d'entorn de `manager_db`:

```
GITHUB_TOKEN=36cf7b2239522186354966ba68d2a0402a97f50e
GITHUB_FATHER_REPO_URL=https://36
   cf7b2239522186354966ba68d2a0402a97f50e@github.com/ChatbotTFG/
```

```
GITHUB_APPLICATION_REPO=ApplicationCode
GITHUB_PROGRAMACIO1_REPO=programacion1
GITHUB_REPOS_ASSIGNATURES='[["programacion1", "Informatica"], ["
    correctorC", "Matematiques"]]'
```

Listing A.3: Variables d'entorn de /manager_db `/lovelaceUB/manager_db/.env`

Les variables d'entorn de `/manager_db` són en referència a noms de repositoris i codis, i estan fetes d'aquesta manera per si es volgues canviar el repositori on es contenen els exercicis que es pugui fer sense entrar a les variables internes del codi. Les variables potencialment sensibles no es censuren ja que s'ha de garantir accés al repositori per fer-hi peticions, per tant no es tracta d'informació tan delicada com als altres dos entorns.

La consideració important és la llista `GITHUB_REPOS_ASSIGNATURES`, la qual ha de contenir el nom del repositori exacte tal com s'anomena al GitHub, sinó no es podran clonar els repositoris. També és important posar-hi el grau i que sigui uniforme per a futures funcionalitats.

Es descriu a continuació l'últim entorn, el del *Rasa Action Server*, ja que `Corrector` i `Rasa` no tenen variables d'entorn.

```
OPENAI_SECRET_KEY = redacted
```

Listing A.4: Variables d'entorn de /rasa/actions `/lovelaceUB/rasa/actions/.env`

Per obtenir una clau d'*OpenAI* per la connexió al model *GPT-3.5-turbo*, s'ha de fer un compte al seu web. Malauradament, no es gratuïta, així que per això s'ha exclós el seu valor de la memòria.

A.3 Execució en Local i *ngrok*

Quan un *webhook* és disparat, ha d'atacar una URL HTTPS, no HTTP, fent que atacar el projecte executant-se en una màquina local sigui, a priori, impossible. Per a canviar-ho, s'usa *ngrok*, que bàsicament proporciona una URL HTTPS que, quan rep una petició, és redirigida a la màquina local que està executant *ngrok*. La comanda per iniciar *ngrok* per les necessitats del projecte és la següent:

```
ngrok http <PORT> --host-header=l·localhost:<PORT>"
```

Seguint l'exemple de `/chatbot` i els valors descrits a les variables d'entorn, la comanda seria `ngrok http 8443 --host-header=l·localhost:8443"`, on estem dient a *ngrok* que redeirigeixi la petició que li arribi a http al port 8443 i al *host localhost* (la màquina local).

Això ens porta a la limitació principal: *ngrok* només permet una connexió simultània i que canvia per execució. La no simultaneïtat implica que els *webhooks* de *push* de GitHub no

poden ser testats simultàniament que el *webhook* de Telegram està actiu (i per tant l'agent rep missatges). La segona és que, per a cada vegada que s'aturi la comanda d'*ngrok*, s'ha de canviar la variable.

A.4 Execució

Per l'execució del projecte amb docker, un cop comprovats que tots els requeriments estan satisfets i els `.env` descrits a l'apartat anterior amb el valor adequat de les variables, s'ha d'executar la comanda al directori arrel del projecte:

```
docker-compose up --build
```

Que construeix els Dockers de tots els mòduls i els aixeca. Per aturar l'execució, s'ha cancel·lar el procés amb ctrl+c, i per abaixar els contenidors d'ha d'executar

```
docker-compose down
```

que eliminarà la xarxa virtual i els contenidors.

Si el projecte es vol executar en local, s'han d'emprar cinc processos, entrant als respectius entorns:

- `Xatbot`: Entorn `/venv_chatbot`. Comanda: `python -u chatbot.py`
- `GestorDB`: Entorn `/venv_managerDB`. Comanda: `gunicorn manager_db:app -c gunicorn.conf.py`
- `Corrector`: Entorn `/venv_managerDB`. Comanda: `gunicorn corrector:app -c gunicorn.conf.py`
- `Rasa`: Entorn `/venv_rasa`. Comanda: `rasa run --enable-api`
- `Rasa`: Entorn `/venv_rasa`. Comanda: `rasa run actions`

Si s'executa el projecte en local, s'ha de canviar l'endpoint de l'`Action Server` al `config.yml` de `/rasa`, ja que al ser un yml, no s'ha aconseguit que capturi la variable de l'entorn de Python.

A.5 Endpoints

Per consultar els *endpoints* de les aplicacions `GestorDB` i `Corrector`, executis en local el servidor i accedeixis a l'URL següent al navegador: `http://MANAGERDB_LOCAL_HOST:MANAGERDB_PORT/docs` i `http://CORRECTOR_LOCAL_HOST:CORRECTOR_PORT/docs` on es poden trobar els *endpoints* d'aquestes dues aplicacions cortesia de FastAPI.

Per els de `Xatbot`, es poden consultar les rutes que s'activen al fitxer `/chatbot/chatbot.py` separats i comentats depenent del mòdul al que apunten. La comunicació entre ells s'ha descrit ja al'apartat 8.2, consultis per més informació.

A.6 Excepcions

El codi contenint el que emmagatzemen les excepcions i la seva definició és a `/manager_db/model/Exceptions.py`

- DatabaseConnectionError: Aquesta excepció es llança quan es produeix un error en intentar establir una connexió amb la base de dades. Juntament amb el missatge d'error, s'emmagatzema un codi d'error opcional (error_code) que proporciona més detalls sobre la causa específica de l'error.
- TableCreationError: Aquesta excepció es produeix quan hi ha un error en intentar crear una taula en la base de dades. El missatge de l'excepció hauria d'incloure informació rellevant sobre l'error i també es pot incloure el nom de la taula que es va intentar crear (table_name).
- TableTruncateError: Es llança aquesta excepció quan es produeix un error al intentar esborrar totes les dades d'una taula (operació de truncat). L'excepció inclou el nom de la taula que es va intentar truncar.
- ItemInsertionError: Aquesta excepció es llança quan hi ha un error al intentar inserir un nou element en una taula. El missatge de l'excepció hauria d'incloure detalls sobre l'error i també es pot incloure el nom de la taula en què es va intentar la inserció.
- AttributesRetrievalError: Aquesta excepció es llança quan es produeix un error en intentar recuperar atributs específics d'una taula. Juntament amb el missatge d'error, s'emmagatzemen el nom de la taula i la llista d'atributs que es van intentar recuperar.
- ItemRetrievalError: Aquesta excepció es produeix quan es produeix un error en intentar recuperar un element de la base de dades. Juntament amb el missatge

d'error, s'emmagatzema l'element que es va intentar recuperar i qualsevol detall adicional sobre l'error.

- ItemRemovalError: Es llança aquesta excepció quan hi ha un error al intentar esborrar un element de la base de dades. Juntament amb el missatge d'error, s'emmagatzemen el nom de la taula, la llista de condicions que defineixen quins elements s'han d'esborrar i els valors corresponents per a aquestes condicions.

- ItemUpdateError: Aquesta excepció es llança quan es produeix un error en intentar actualitzar un element de la base de dades. Aquesta excepció emmagatzema el missatge d'error, el nom de la taula, la llista d'atributs que s'han d'actualitzar, els nous valors per a aquests atributs, i la llista de condicions i valors que defineixen quins elements s'han d'actualitzar.

- ItemNotFoundError: Es llança aquesta excepció quan es busca un element específic en la base de dades i aquest no es troba. Juntament amb el missatge d'error, s'emmagatzemen el nom de la taula, la llista de condicions que defineixen quin és l'element buscada i els valors corresponents per a aquestes condicions.