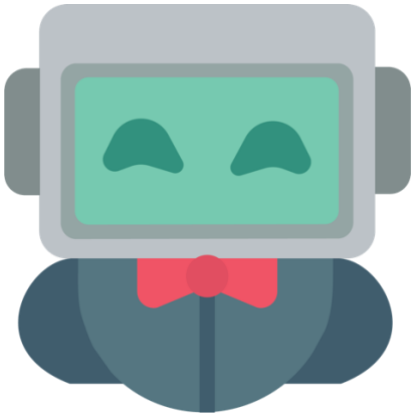
Theatro: Un Chatbot de pel·lícules i series implementat amb DialogFlow , Neo4J i GraphQL

Biel Carpi · Victor Xirau · Rafael Morera



Theatro

Els ChatBots de conversa contemporanis són fàcils d'utilitzar i tenen la capacitat de simular converses humanes. Tanmateix, no poden avaluar grans conjunts de dades exhaustius per donar una resposta a l'usuari. En canvi, disposem de moltes APIs gratuïtes que contenen aquestes dades i poden oferir-les donat un seguit de paràmetres. En esta investigació vam dissenyar un servidor capaç de escoltar peticions de DialogFlow i processar-les emprant una base de dades de Graph per millorar l'experiència d'usuari i oferir un volum de informació superior al habitual al client. Mitjançant Google Dialogflow analitzem i GraphQL tradueix la query Cypher per peticionar la nostre base de dades Neo4J, aquesta analitza i proporciona una resposta precisa mitjançant un conjunt de dades complet i simula una conversa sembla a un humà. Utilitzem la base de dades per oferir respostes precises mitjançant un conjunt de dades de resposta i Google Dialogflow per simular interaccions humanes.

Termes Generals: chatbot, IA, movies, conversation, dialogflow, movies, tv-shows

---

Introducció

Tots nosaltres ens hem trobat en la situació d'estar navegant durant hores per Netflix, HBO, Disney+ cercant una pel·lícula o sèrie per veure, sense èxit. En quant es vol cercar informació sobre pel·lícules, sobre els actors o fer recomanacions informades, hem de passar a fer ús de pàgines com IMDB o Rotten Tomatoes, que son bastant complicades de emprar per un usuari mig. La nostres solució és simple: crear un chatbot que et recomana pel·lícules i series i proveeix informació sobre elles al usuari. Aquest es presentarà a través de la interfície d'un mòbil i es dirà Theatro.

Els ChatBots son programes que estimulen parlar com un humà. Sovint estan dissenyats amb un objectiu en ment – en el nostre cas, donar informació sobre contingut audiovisual ja siguin pel·lícules o series. Però és important saber que hi ha un seguit de components necessaris per construir un ChatBot:

1. *Processat de Llenguatge Natural* – separar les frases de input entre les parts que la formen
2. *Generació de Quèries* – traduir les frases processades a quèries valides
3. *Gestió de Context* – Recordar les respostes a quèries anteriors per contestar-ne futures de manera intel·ligent
4. *Coneixement del Domini* – Desar el coneixement sobre el domini en que el ChatBot se centra
5. *Generació de Llenguatge Natural* – Donar personalitat al ChatBot

Trobar la bona combinació d'eines que permetin construir un bon chatbot pot suposar un repte. Clarament existeixen moltes plataformes basades en Chatbots. Aquestes solen ser plataformes amb un "bot-as-a-service" on es pot construir, adaptar i desplegar un servei al cloud. Algunes d'aquestes opcions inclouen BotEngine,

Microsoft Bot Framework, DialogFlow o IBM Cloud Watson. Totes elles proveeixen un set de funcionalitats, integracions i diferents nivells de usabilitat.

A més, els ChatBots sovint requereixen un backend amb una base de dades, on tornarem a afrontar una tria. Les opcions van des de les bases de dades estàndard SQL, amb una estructura poc compatible amb llenguatge natural, o d'altres més accessibles com les bases de dades NoSQL. També és pot triar la alternativa de tenir un backend basat en Graphs, com nosaltres. Les quèries en una base de dades basades en graph sovint s'assemblen a les connexions presents al llenguatge natural, el que les fa ideals per construir ChatBots.

Pel que fa a la eina de processat de llenguatge natural, tot i que alguns dels autors del article havien treballat anteriorment amb OpenNLP i altres eines pel processat, Dialogflow les guanya – per molt – tant en termes de precisió com d'usabilitat. Dialogflow proveeix eines per mantenir els contextos i et permet afegir respostes personalitzades quan les quèries fallen.

Aquestes respostes han de tenir un fonament amb dades reals, pel que caldrà crear un servidor entremig que gestioni el missatge de DialogFlow, el peticioni a la BBDD, i contesti a DialogFlow novament. Es vol tenir el màxim coneixement possible de les pel·lícules, pel que emprarem les dades d'IMDb, la font més popular del món de contingut de pel·lícules, televisió i celebritats

Contingut

Antecedents

Theatro és el resultat de un munt de projectes previs ja sigui per part nostre o que hem tingut la sort de trobar per internet i servir-nos d'inspiració. Alguns exemples d'aquests antecedents, i com ens han servit com a mesura de contrast per veure com de lluny hem arribat son la primera implementació d'aquest mateix chatbot: TV-Chatbot. Aquesta primera versió era molt rudimentària, basada amb regles i amb un domini molt limitat donat que només era capaç de recomanar pel·lícules. En recomanava en funció del gènere o generes que es demanessin, i de les plataformes de streaming on es pogués trobar aquell contingut. L'aspecte més complexe d'aquella implementació era clarament el tractament del llenguatge natural. Aquest era un seguit de regexs que es comparaven amb el input, però que només eren capaces de contestar a un seguit de preguntes fetes per adelantat, i qualsevol cosa que se sortís d'això, rebria una resposta "estàndard" de "no se que m'estàs dient".

Hem trobat també gent que ha treballat amb DialogFlow anteriorment que ens ha servit molt per poder trobar exemples de com crear certes instàncies, o quin tipus de informació incloure a les entities, etc. Però cap tenia una arquitectura tan complexa i escalable com la nostre, totes elles tenien un plantejament més aviat simple, d'exemple, per poder il·lustrar el tutorial o eina que estiguéssin ensenyan en aquell moment i poc més.

Presentació de la Proposta

Pel que fa a la nostre proposta, dividirem l'explicació en 3 apartats, donat que tenen una lògica ben diferenciada i una raó de ser també molt diferent. Aquests tres apartats seran DialogFlow, Servidor i Base de Dades. Part de la complicació del chatbot deriva de com gestionem aquests 3 aspectes i el procés que hem seguit per acabar tenint-ho com a la Figura 1.

Podem veure com aquí s'interacciona amb Telegram, però aquest parlarà amb DialogFlow qui també peticionarà al seu webhook, hostejat a Heroku per que sigui accessible públicament. Aquest servidor el que farà és gestionar la petició de DialogFlow, convertir-la a GraphQL i peticionar la nostre base de dades Neo4J hostejada a una VM a GoogleCloud.

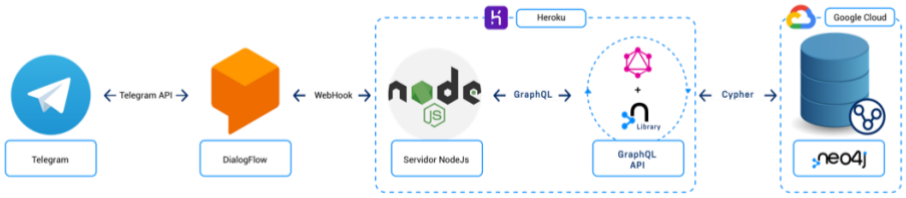


Figura 1: Arquitectura de Theatro

DialogFlow

Per la creació del DialogFlow s'han seguit uns passos molt senzills, primerament s'ha creat el agent de pel·lícules a la plataforma:

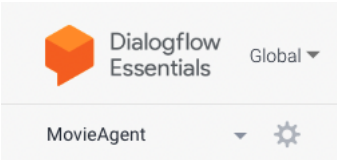


Figura 2: Movie Agent

Aquest és essencialment el vostre ChatBot. Podem configurar un agent preexistent o crear-ne un a través de la consola. Com hem fet nosaltres. L'hem anomenat MovieAgent.

Dins d'aquest trobem dos caps importants. Aquests seran els que ens permetran veritablement interactuar amb el chatbot e interpretar amb llenguatge natural el que ens estan dient.

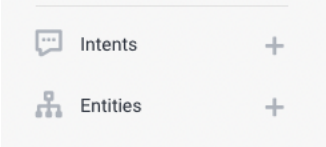


Figura 3: Intents & Entities

En primer lloc tenim els Intents. Aquests capten la intención d'un usuari que interactua amb el nostre agent. Qualsevol cosa que l'usuari escrigui/parli amb l'Agent, s'assigna a una intenció. Per defecte en tenim dues de "Fallback". A més d'aquests n'hem creat 5 més:

Figura 4: Els nostres Intents

Es pot veure com s'han creat intents per cada una de les funcionalitats del chatbot. Tenim un intent per el processat d'una recomanació, quan ens demani recomanar-li alguna pel·lícula. Tenim un intent dedicat a la creació del bot, a ajuda, a informació i a agraïments.

Cada un d'aquests ha estat entrenat amb un munt d'exemples d'input per part de l'usuari, i d'entities.

Què son aquestes Entities?. Si pensem en el nostre MovieAgent, quan l'usuari diu: "Dona'm informació sobre Jobs", això hauria de dir a l'agent que necessito informació sobre la pel·lícula: Jobs. Però com sap això, com és capaç de analitzar amb aquest detall la fase? Bé, per cada expressió d'usuari assignada a una intenció, el nostre agent necessita informació. Aquesta la obté amb ajuda de les Entitats. Cada expressio d'usuari d'una intenció pot tenir una entitat que defineix la cosa, de la qual l'usuari necessita informació.

Figura 5: Les nostres Entities

Aquestes son, categoria de la pel·lícula, son els tags pels quals pot identificar-se una pel·lícula, ja sigui gènere o simplement adjectius que la descriuen. Content-type, si volem pel·lícules, series, curts, etc. Hem declar una de movie, per que sigui capaç d'entendre noms de pel·lícules i series, un de roles per tots els possibles rols d'una producció, per que es pugui cercar amb aquesta restricció, i finalment una entity dinàmica de cerca, que combina les anteriors entre sí.

És important mencionar com, a més de les nostres entities, DialogFlow té les seves entities pròpies, com ara la detecció de números o noms propis.

Havent definit tot això, el que va caldre és entrenar amb molts exemples a aquesta IA, per que pogués saber quin tipus de missatges esperar-se. Com no és d'estranyar, Google és molt potent, i només cal posar un seguit d'exemples i ja és capaç d'entendre'n de nous amb els que no se l'hagi entrenat.

A continuació, calia vincular la pròpia interfície gràfica. El propi DialogFlow ofereix un seguit de integracions directes amb un munt d'eines com ara Slack, Line, Skype o Telegram. En el nostre cas, vam voler triar Telegram donat que creiem que és la més potent de totes elles, i ja l'havíem emprat anteriorment per l'altre Chatbot i vam creure adient fer-lo servir per aquest.

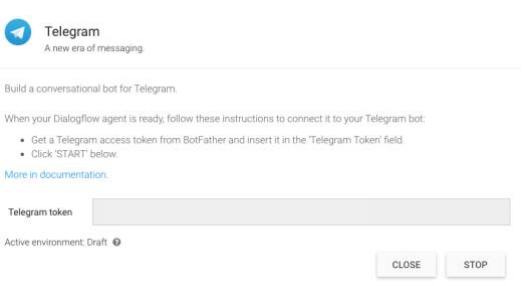


Figura 6: Configuració Telegram

Per últim, vam afegir un extra que ofereix DialogFlow que és la possibilitat de tenir converses trivials amb el Chatbot. Preguntar-li si està feliç, dir-li que estàs trist, o que no has entès la resposta/pregunta realitzada. És el que DialogFlow anomena "Small Talk", una funcionalitat que permet que de manera intel·ligent, i novament amb una IA de Google ja entrenada Theatro se senti més humà, i no tingui moments on no sàpiga que fer.

Arribats a aquest punt el que ens caldrà però, és enllaçar els intents amb el servidor que s'explicarà més endavant. Per fer-ho, es pot anar a la part de "Fullfilment" de la consola, i configurar el que es coneix com un WebHook, un mètode per augmentar o alterar el comportament d'una pàgina web o aplicació web amb devolucions de callbacks personalitzades.

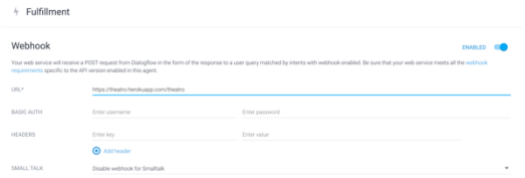


Figura 7: Fullfilment Webhook

Els intents tenen un apartat de respostes, on es poden configurar un seguit de respostes estàtiques per missatges d'aquest tipus.

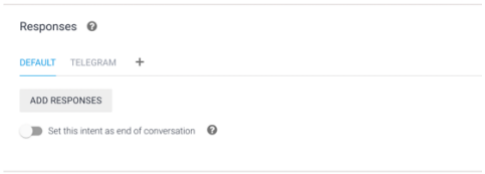


Figura 8: Intent Responses

Donat que no volem respostes estàtiques, més aviat al contrari. Respostes que es generin de manera dinàmica parlant amb la nostre base de dades, caldrà posar com a response el nostre WebHook, per fer-ho anirem a la part de FullFilment dins d'un Intent i li direm que sí a fer servir el WebHook com a resposta.

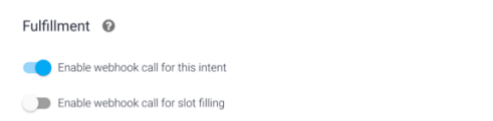


Figura 9: Fullfilment Responses

Semblaria que ja ho tenim. Això el que farà serà peticionar el webhook establert mitjançant una petició post amb el payload al body. Aquest payload serà la informació detallada extreta del missatge.

Figura 10: Detalls d'un missatge

Aquesta informació detallada té un format JSON, i podem veure com a queryText tenim la pregunta original, a paràmetres els valors importants d'aquesta frase, en aquest cas el nom de la pel·lícula, i en aquest cas el servidor ens ha pogut respondre, trobarem la resposta a fulfillmentMessages.

Base de Dades

De les 5 parts que té el desenvolupament d'un chatbot, introduïdes al inici d'aquest article, hem cobert ja la primera. Ara fem un salt a la 4a, on fem referència al Coneixement del Domini.

Per la primera implementació vem emprar una API a la que peticionàvem la recomanació, i era aquesta la que tenia la base de dades com a tal. En aquesta implementació vam voler tenir el màxim de llibertat possible pel que no volíem dependre de cap API. Sí vam realitzar una breu cerca per veure si hi havia APIs que complissin amb els nostres requisits de manera gratuïta, però era un objectiu poc realista.

Per la generació de la nostre base de dades pròpia es va voler emprar el data set de dades més gran possible, però que alhora ens donés una mínima relació entre les dades per poder fer quèries complexes, i se'ns donés un volum

d'informació ampli. És per això que, després de passar moltes hores investigant múltiples datasets, vam optar per emprar les dades de la pàgina web IMDB. Però fer-ho no va ser tant evident com podria sembla en una primera instància.

Hi ha subconjunts de dades d'IMDB disponibles per accedir als clients per a ús personal i no comercial. Cada conjunt de dades està inclòs en un fitxer amb format de valors separats per tabulacions (TSV) en UTF-8. La primera línia de cada fitxer conté capçaleres que descriuen què hi ha a cada columna. Un "\N" s'utilitza per indicar que falta un camp en particular o és nul per a aquest títol/nom. Tot i així, per la correcte importació d'aquestes dades a qualsevol base de dades que acabem emprant caldria convertir-los a un format més universal com ara CSV. Per fer-ho, vam realitzar un seguit de scripts en Python que netegen els inputs, i els converteixen al format desitjat. Aquestes dades ja ens eren útils, però no havíem decidit encara amb quina arquitectura faríem la base de dades.

Tal i com es comenta a la introducció, hi ha tres models principals des dels que triat: SQL, NoSQL i Graph. Tot i ja saber quin és el resultat que hem triat donat que ho hem mencionat anteriorment, esmenarem una mica el procés de prova i error que s'ha seguit per decidir finalment aquesta arquitectura.

Primerament, vam optar per una aproximació SQL. El motiu era senzill: conservar les relacions, i que era la tecnologia que ens era més familiar per la experiència que tenim. Vam decidir emprar PostgreSQL i procedir a la creació de totes les taules amb els seus respectius atributs i relacions. La importació va ser un procés que va trigar molt més del que esperàvem, però va ser satisfactori. El problema era que tal i com estaven creades les dades de IMDB, les quèries més simples ja requerien de un seguit de subqueries o joins que feien del procés de creació de la query força complicat, i tenien una execució molt lenta.

Figura 11: PostgreSQL

Veient aquestes limitacions, va ser quan vam decidir provar un altre aproximació a la base de dades, però fent servir el que ja teníem: fer un schema de Graph amb AgensGraph. AgensGraph és una base de dades de grafs multi-model de nova generació per a PostgreSQL. Ofereix l'entorn d'anàlisi de grafs per a dades altament connectades en què els usuaris poden escriure, editar i executar consultes SQL i Cypher alhora.

Figura 12: Agens i Cypher

Per duu a terme aquests canvis, vam fer un seguit de modificacions a la base de dades SQL, com ara fer views per intentar ajuntar algunes de les taules relació i evitar tantes subqueries, i com és evident, crear tots els nodes i relacions de la base de dades amb quèries Agens.

I funciona! Ja ho teníem! Ja teníem una base de dades de grafs, corrents damunt d'una base de dades SQL. Ara teníem només 2 problemes.

1. La base de dades estava en local, i per tant qualsevol servidor hostejat de manera remota no podria interaccionar amb ella.
2. Les quèries ara s'havien de fer amb Cypher. Aquestes eren més completes que les quèries SQL, i ens permetien un cert nivell d'abstracció, però encara suposaven un problema de cara a crear-les de manera dinàmica.

Passem a la terca implementació. Vistes aquestes limitacions vam passar a solucionar la primera, i la solució va ser senzilla: crear una instància RDS a Amazon Web Services, i copiar totes les dades que tenim en local, a la base

de dades remota. Això va funcionar correctament, i teníem la base de dades SQL a AWS, però en intentar copiar Agens va sorgir, novament, un altre problema: Amazon Web Services no es treballa amb AgensGraph. Tot i que era una eina que funcionava damunt de PostgreSQL, i teníem totes les esperances depositades en que sí funcionés, no va ser el cas.

Figura 13: Amazon RDS

Davant d'això, com es pot veure, no vam aturar-nos, i vam fer un redisseny més a la arquitectura de la base de dades, intentant conservar la feina feta fins ara. Aquesta passava per intentar buscar un substitutiu de AgensGraph, un que sí pogués funcionar conjuntament amb PostgreSQL. Després d'una exhaustiva cerca es va trobar: Hasura. Hasura us ofereix GraphQL i API REST instantànies en fonts de dades noves i existents, però el que més ens cridava l'atenció de Hasura és poder emprar GraphQL damunt de la nostre base de dades ja existent.

GraphQL és un llenguatge de consulta flexible que utilitza un sistema de tipus per retornar dades de manera eficient amb consultes dinàmiques. SQL és un estàndard de llenguatge més antic i més adoptat utilitzat específicament per als sistemes de bases de dades tabulars/relacionals. Com a tal, ens permetia un altre nivell d'abstracció, superior al Cypher d'Agens, on podíem fer quèries molt més simples i ràpides però obtenir els mateixos resultats.

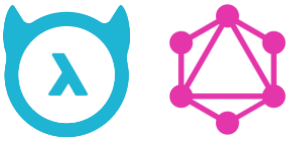


Figura 14: Hasura i GraphQL

Arribats a aquest punt, tenim una base de dades PostgreSQL a una instància de AWS RDS i un endpoint de Hasura vinculat amb aquesta instància al que faríem peticions GraphQL directament. I semblaria que ja ho tindríem! Ja hauríem arribat al disseny definitiu pel nostre ChatBot, que ens permetria la major versatilitat i velocitat amb les dades. Però per l'extensió d'aquest article pot veure's com no és el cas. Teniem, novament, dos problemes nous:

1. Resulta que els datasets que creiem haver important correctament, no estaven ben importats. Va ser en aquest punt, on emprant quèries més complicades vam adonar-nos que hi havia relacions que no estaven ben establertes, dades que faltaven o que simplement eren incoherents, etc.
2. Les quèries trigaven **molt** en resoldre's. Vam fer un munt d'optimitzacions tals com indexs a les bases de dades, vistes, etc però cap va funcionar. El problema s'originava per l'arquitectura en sí. Hasura tenia els servidors hostejats a Estats Units, mentre que la nostre instància de AWS estava a Europa, i això, juntament amb que el nostre volum de dades era enorme, i erroni, feia que triguessin molt a finalitzar.

Que es fa davant d'aquesta situació? Tornar a començar. Amb forces renovades, però tornar a replantejar tota la base de dades, aquest cop amb les coses clares: Voliem una arquitectura de Grafs i volíem evitar el ús de Cypher per fer les quèries, pel que seguíem volent emprar GraphQL per abstreure'ns del ús de Cypher.

El primer pas seguit és eliminar PostgreSQL de la equació, i replantejar la importació de dades de 0, directament a una base de dades basada en Grafs, per evitar passar per SQL i el que suposa a nivell de temps d'execució de queries. Triar la tecnologia no és pas trivial, vam fer una cerca important per decidir quines hi havia al mercat, quines ens oferien el major nombre de funcionalitats de manera gratuïta, i quines podien adaptar-se millor al tipus i volum de dades que tenim.

Hi ha un top 3 molt diferenciat: ArangoDB, Neo4J i DGraph. Cada una d'aquestes té les seves avantatges i desavantatges, i la veritat és que la tria final no ha estat del tot imparcial. D'aquestes 3 el clar guanyador per prestacions, capacitats i potència és DGraph. És la base de dades GraphQL més avançada del món amb un backend gràfic. La base de

dades de gràfics número u a GitHub i més de 500.000 descàrregues cada mes, Dgraph està dissenyada per al rendiment i l'escalabilitat. És la base de dades emprada per empreses com Intel, Cisco, Dell i centenars més que estan llistades a la seva web. Però tot i així no és la que hem triat. ArangoDB també està molt ben valorada, però no gestionant tant bé com DGraph els volums de dades enormes i no té un format tant "dinàmic" com ens interessaria donat que funciona amb un sistema de JSONs clau-valor. Per últim tenim Neo4J, descrit pels seus desenvolupadors com una base de dades transaccional compatible amb ACID amb emmagatzematge i processament de gràfics natius. És potser de les més conegudes per la seva longevitat, i ha estat la eina triada per que ens és familiar. Sabem que no és dels millors arguments, però no és una mala tecnologia, està en el top 3 de bases de dades de grafs, i ja hi hem treballat amb anterioritat a la universitat, pel que estem familiaritzats amb com funciona.

Figura 15: ArangoDB, Neo4J i DGraph

Decidit el model, vam procedir a la creació dels nodes i la seva importació. Amb les dades netes gràcies als primers apartats del procés, vam realitzar un seguit de quèries Cypher per importar les dades des dels csvs a una base de dades local de Neo4J. Un cop fet això, vam provar la base de dades amb quèries cypher per comprovar que funcionava, i que ara sí teníem les relacions que no teníem des de PostgreSQL, i funcionava tot perfecte.

Quedaven pocs passos per acabar. Seguiem volen abstreure'ns de Cypher, pel que vam emprar una llibreria anomenada Apollo que ens permet tornar a fer ús de GraphQL per fer les quèries a la base de dades, i aquestes es tradueixen a Cypher. Per últim, faltava fer el deploy de la base de dades. Per fer-ho, s'ha creat una instància d'una màquina virtual a Google Cloud que es peticiona des del servidor.

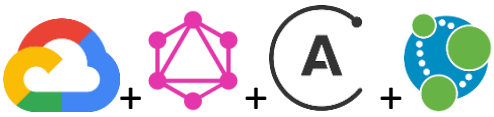


Figura 16: Base de Dades Final

Servidor

A aquestes alçades es té un coneixement extens dels dos extrems del nostre chatbot. De la part de frontent, el Telegram i DialogFlow, encarregats de enviar i processar el text del usuari, i extreure'n informació rellevant per posteriorment emprar-la amb la base de dades. També tenim clar el procediment seguit per la realització de la base de dades i la arquitectura final emprada. El que no queda clar encara és com es comuniquen entre sí aquests dos extrems, ho fan gràcies al nostre servidor.

Si recordem d'apartats anteriors, DialogFlow peticionarà al WebHook mitjançant una petició POST. Pel que des del nostre servidor, haurem de rebre aquesta petició, tractar-la, i contestar a DialogFlow.

Per fer-ho hem desenvolupat un servidor amb NodeJS que fa precisament això. Escolta peticions de tipus POST a un endpoint concret. En rebre'n una, diferenciem el tipus de intent que s'ha realitzat, convertim les dades a GraphQL amb un seguit de condicions i peticionem a la base de dades per que aquesta ens retorni el resultat desitjat.

Figura 17: Node JS

Aquestes dades es retornen amb el format demanat per DialogFlow, per que aquest contesti al client de Telegram. Inicialment aquest format podia ser un format de text simple, sense cap altre detall. Més endavant ens vam adonar de la capacitat de DialogFlow de treballar amb el que ells anomenen "Cards", Targetes. Aquestes poden tenir un format concret, que s'omple dinàmicament des del client per contestar a Telegram amb un format més interessant que no pas un texte simple.

Experimentació

Amb un Chatbot funcional i domini com el nostre hi ha una infinitat d'exemples que es poden fer, i convidem a qualsevol a fer-lo anar visitant aquest enllaç i fent les preguntes que cregui convenient!

Exemples

Per aquesta part el que farem serà posar exemples en dificultat incremental, per poder anar veient com respon a cada una de les situacions, i com és l'output.

Primerament farem un exemple senzill, on l'usuari acaba de connectar-se amb el chatbot i es presenta e interactua amb ell.

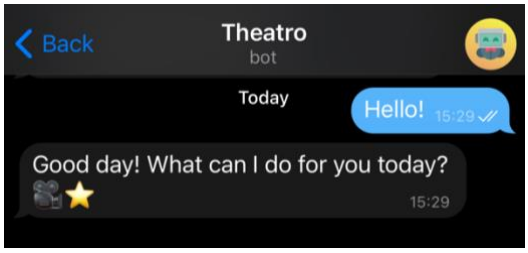


Figura 18: Exemple benvinguda

A més, se li podrà demanar ajuda, o informació sobre com fer-lo anar. Podem veure com en aquest cas, la resposta sí és una resposta pre-definida per nosaltres, on indiquem concretament què pot preguntar-nos.

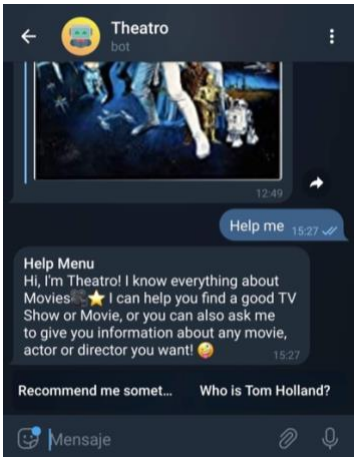


Figura 19: Exemple help

Començant a entrar en matèria, podem fer un seguit de consultes preguntant informació sobre una persona o pel·lícula concretes:

Figura 20: Exemple informació Star Wars

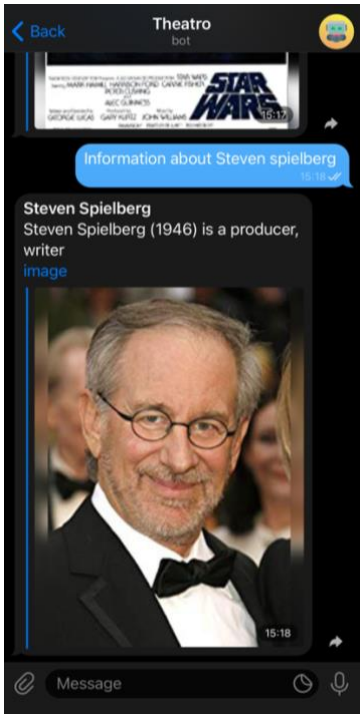


Figura 21: Exemple Informació Steven Spielberg

En aquests exemples es dona ja una informació bastant significativa de les dades demanades.

Finalment, l'exemple potser més clar del funcionament inicial del chatbot, és el de demanar una recomanació.

Figura 24: Recomanació de película

Temps de Execució

Per la mesura del temps d'execució, hem fet dues mesures, la primera mesura s'ha fet des del servidor, per saber quan triga la base de dades en contestar a un seguit de quèries. La segona en canvi, ens la dona el mateix DialogFlow, que és el temps total per resoldre la petició.

És interessant tenir en compte que la nostre base de dades pesa 20Gb dona que té un total de 25 Millions de nodes i un total de 39,42 Millions de relacions entre elles.

Les peticions realitzades han estat principalment tres, que s'han fet repetides vegades per poder assegurar que les dades son coherents. Primerament hem mesurat una petició de informació, on es demana informació sobre la pel·lícula de Star Wars.

"Show me Information about Star Wars"

DIALOG FLOW

Per aquestes hem emprat la resposta de DialogFlow que hem suposat mesura el temps tota de tota la transacció. No podem mesurar el temps de Telegram donat que no tenim manera de fer-ho pel que aquesta és la aproximació més fidel al temps de latència del chatbot.

Nº Ej	Tiempo	Nº Ej	Tiempo
1	2588ms	6	1602ms
2	2439ms	7	2100ms
3	1428ms	8	2349ms
4	2179ms	9	1953ms
5	1980ms	10	1973ms

Figura 26: Temps execució Dialog Flow Q1

Com podem veure els resultats son molt propers als 2s, no és un temps genial, però és curiós per que la majoria dels problemes no deriven de la base de dades, com veurem a continuació.

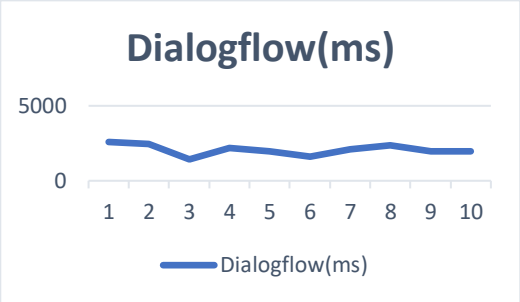


Figura 27: Gràfica execució Dialog Flow Q1

BASE DE DADES

Nº Ej	Tiempo	Nº Ej	Tiempo
1	315,87ms	6	167,42ms
2	176,13ms	7	174,48ms
3	168,49ms	8	167,22ms
4	148,98ms	9	172,37
5	176,55ms	10	217,86ms

Figura 28: Temps execució Base de Dades Q1

La base de dades té un temps molt raonable! Veiem que la mitjana està al voltant dels 188ms de temps de resposta. Considerem un temps molt raonable, novament tenint en compte el volum de les dades. En aquest cas la query està cercant pel·lícules amb el nom "Star Wars" i recuperant, a partir del id d'aquesta peli, la informació de la mateixa.

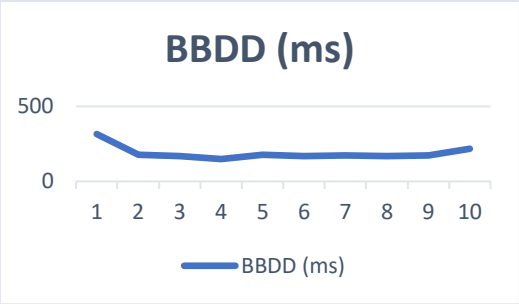


Figura 29: Gràfica execució Base de Dades Q1

Si podem veure en canvi, com a la gràfica de la base de dades hi ha un pic inicial, i després les demés mesures son significativament més baixes. Això és degut a que Apollo implementa una caché interna de les peticions GraphQL per optimitzar el servidor.

A continuació hem realitzat una petició on es demanés informació sobre una entitat ja sigui el nom d'un actor o d'una pel·lícula o sèrie. Seguim el mateix format que al apartat anterior on mesurem primer el temps total, el de DialogFlow, i el temps de la base de dades mesurat des del servidor.

"Give me Info about Anthony Michael Hall"

DIALOG FLOW

Novament, veiem que el temps de DialogFlow no dista gaire dels 2s, per tant és bo saber que canviant de query no obtenim resultats molt dispars. El que sí veiem també és que la caché que comentàvem que estava present a la base de dades, es reflecteix també aquí, on veiem que hi ha una primera petició de gairebé 3s, i les demés estan més aviat al voltant dels 1.5s.

Nº Ej	Tiempo	Nº Ej	Tiempo
1	2944ms	6	1427ms
2	1495ms	7	1865ms
3	1162ms	8	1204ms
4	1085ms	9	1195ms
5	1623ms	10	1458ms

Figura 30: Temps execució Dialog Flow Q2

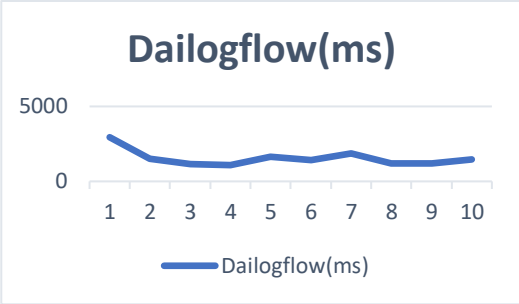


Figura 31: Gràfica execució Dialog Flow Q1

Novament es produeix el mateix fenòmen a la base de dades. Les quèries triguen una mitjana de 133ms en executar-se, només 50ms menys que la anterior query, pel que son molt properes en temps.

BASE DE DADES

Nº Ej	Tiempo	Nº Ej	Tiempo
1	271,75ms	6	122,45ms
2	119,67ms	7	133,12ms
3	111,80ms	8	97,87ms
4	123,85ms	9	120,38ms
5	121,90ms	10	116,98ms

Figura 32: Temps execució Base de Dades Q2

També veiem clarament com la caché està funcionant, donat que tenim el temps de la primera petició bastant elevat, i el temps de totes les demés és poc menys de la meitat.

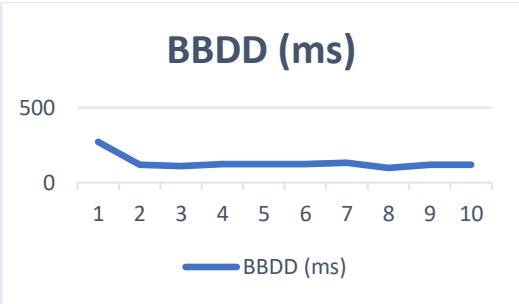


Figura 33: Gràfica execució Base de Dades Q2

A continuació hem realitzat una query més complexe, on es demana que se'ns recomani una pel·lícula. Aquesta sí té més complicació donat que ha de triar seguint un criteri concret d'entre les pel·lícules de la base de dades.

“ Recommend me Something”

DIALOG FLOW

Nº Ej	Tiempo	Nº Ej	Tiempo
1	2247ms	6	2066ms
2	1965ms	7	1963ms
3	2172ms	8	2107ms
4	2484ms	9	2205ms
5	2035ms	10	2668ms

Figura 34: Temps execució Dialog Flow Q3

Novament tornem a veure un comportament similar per part de Dialog Flow, on les quèries tornen a trigar al voltant de X però sí veiem que es més [RAPID O LENT] que les anteriors, segurament donat a la naturalesa de la query realitzada.

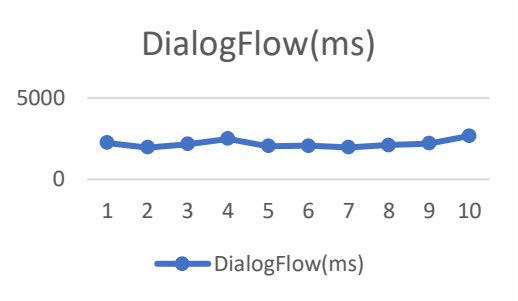


Figura 35: Gràfica execució Dialog Flow Q3

Si observem les dades de la base de dades veiem que aquestes tenen uns resultats, novament molt similars als apartats anteriors. On veiem una primera query que triga més que les temés.

BASE DE DADES

Nº Ej	Tiempo	Nº Ej	Tiempo
1	315,82ms	6	191,63ms
2	206,81ms	7	192,93ms
3	199,02ms	8	180,3ms
4	206,80ms	9	195,47ms
5	204,12ms	10	208,36ms

Figura 36: Temps execució Base de Dades Q3

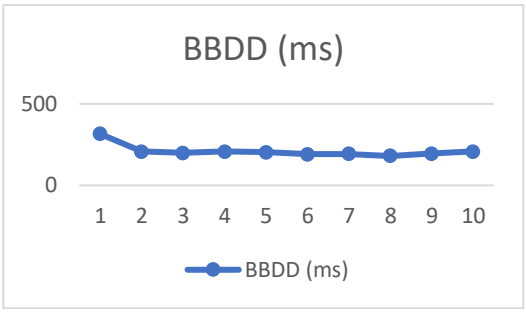


Figura 37: Gràfica execució Base de Dades Q3

Estem força satisfets amb el temps de resolució que té Theatro. Triga fins a un total de 3s en el pitjor dels casos en respondre quèries complexes, amb una base de dades enorme. Ens dona confiança de saber que si volguéssim escalar la base de dades el dia de demà, podríem sense problema. Sobretot sàpiguen que encara podrien fer-se més optimitzacions o relacions per facilitar la cerca d’alguns paràmetres.

Discussió

Com hem vist, fins ara hem treballat amb tecnologies molt diverses que ens han permès aprendre i enfocar el chatbot de maneres molt diferents. La única constant real al llarg de tot el projecte ha estat el ús de DialogFlow ja que al inici es va decidir que es faria amb aquesta.

El resultat final ha estat un chatbot amb una arquitectura molt robusta. Tant és així que es podria reciclar tota la part de backend per un altre projecte sense problema! És completament dinàmica i molt potent.

Després d’haver fet l’anàlisi temporal de la execució de les quèries i de la comprovació del seu correcte funcionament, podem dir que estem molt contents amb els temps de latència i de execució de tot plegat. Era una preocupació que teníem quan vam decidir afrontar un banc de dades tan gran com el que hem emprat, però ha resultat que les solucions que hem plantejat han servit per mitigar al màxim possible aquella preocupació.

Clarament comparat amb el nostre anterior Chatbot, és molt més potent. Tot i així, sí que

hi ha casuístiques que no s'han contemplat que sí contemplàvem al anterior chatbot. Hem perdut la possibilitat de peticionar de plataformes de streaming concretes, donat que no hi son al data set, o de concatenar molts generes entre sí, que donat que ho vam fer hardcoded a la primera implementació sí funcionava però ara al ser dinàmic ens ha suposat més aviat un problema.

Conclusions i Línies de Futur

Fins ara cap dels membres del grup havia treballat mai amb la enorme majoria de tecnologies emprades per aquest projecte. El plantejament inicial, a la versió basada amb regles plantejada anteriorment a l'assignatura de Sistemes Basats en el Coneixement, vam poder fer una primera aproximació al que seria el processat del llenguatge natural e intentar que aquest resulti en una resposta per part del servidor el més clara possible I amb la informació més detallada possible. No teníem base de dades, aquestes venien d'una api externa, i el processat del llenguatge era molt rudimentari i limitat.

Theatro és diferent. Amb Theatro hem pogut posar en pràctica un ventall d'eines i coneixements molt gran. Ens hi hem barallat per decidir quina tecnologia fèiem servir de "frontend" per poder fer el processat del llenguatge natural. Inicialment vam voler implementar la nostre pròpia IA, el que ens va portar a fer una bona recerca en aquell aspecte, aquesta va derivar en el ús d'eines com DialogFlow per deixar aquell aspecte del Chatbot a un servei de tercers molt més potent del que haguéssim pogut fer nosaltres. Això sí, la nostre ambició seguia sense estar satisfeta.

És per això que vam voler fer que les dades amb les que treballés el DialogFlow fossin el més robustes i ràpides possibles. Per fer-ho, tal i com hem detallat a la nostre proposta, vam provar un ventall enorme de tecnologies que ens vam permetre aprendre un munt, portant-nos finalment a emprar Neo4J com a eina per la base de dades i GraphQL com a "Middleware" per realitzar les quèries.

No és el primer cop que treballem amb bases de dades basades amb graphs però sí el primer que ho fem amb tant de detall, i amb una base de dades tan gran.

Ha estat un procés d'autoaprenentatge enorme i de descobriment d'habilitats i eines que no sabíem que teníem. Però havent fet menció a la nostre ambició, un podria creure que aquesta ha estat satisfeta, però no és el cas.

Tenim moltes idees per línies de futur, que no hem pogut implementar per manca de temps però que estaven a la versió inicial de l'entrega. Aquestes passen per una millora estètica dels missatges presentats, per obrir més el ventall de quèries que es poden realitzar, donat que tot i ser molt ampli ara mateix hi ha dades que no estem preparats a donar, com ara el link del tràiler d'una pel·lícula, o preguntes niades o similars. Pel que un aspecte que sí ens agradaria millorar seria la expansió del data set. Som optimistes en aquest aspecte donat que hem emprat un data set universalment conegut com és el de IMDB. Pel que segur que podríem trobar dades compatibles, amb els ids compartits, per afegir. Dades com ara plataformes de streaming on trobar el contingut, reviews dels usuaris més allà de les realitzades a imdb pels crítics, etc.

Per la comprensió d'aquestes també ens hagués agradat afegir d'una forma més robusta la concatenació de condicions amb condicionals com "or" i "and" donat que ara mateix esta fet d'una manera més rudimentària a la part de servidor, però tenint en compte que podem emprar DialogFlow per tasques com aquesta, estem convençuts que podríem acabar obtenint resultats molt superiors als actuals.

Agraïments

Els autors els agradaria esmenar i agrair el suport ofert per la professora Elisabet Golobardes Ribé de La Salle. Els agradaria també agrair l'ajut dels becaris de l'assignatura de Sistemes Basats en el Coneixement, Martí Ejarque i Alejandro Moñux. A la institució de La Salle Campus

Barcelona per proporcionar els recursos, eines i coneixements necessaris per tenir un punt de partida sòlid. També volen esmenar la enorme importància de la disponibilitat gratuïta d'eines com DialogFlow oferta per Google i l'eina Heroku oferta per SalesForce.

Referències

Datt Mamgain, D., 2021. *Dialogflow vs Rasa—Which One to Choose?*. [online] Medium. Available at: <https://chatbotslife.com/dialogflow-vs-rasa-which-one-to-choose-206fb98b0e90> [Accessed 23 May 2022].

Datt Mamgain, D., 2020. *NLP Libraries for Node.js and JavaScript*. [online] Medium. Available at: <https://chatbotsjournal.com/nlp-libraries-for-node-js-and-javascript-1e7d62e4fbfe> [Accessed 23 May 2022].

Wouters, J., 2019. *Dialogflow Tutorial for Beginners ([yoast_year])* | *Chatimize*. [online] Chatimize. Available at: <https://chatimize.com/dialogflow-tutorial/> [Accessed 24 May 2022].

Hasura.io. 2022. *Getting started* | *Hasura GraphQL Docs*. [online] Available at: <https://hasura.io/docs/latest/graphql/core/getting-started/index/> [Accessed 24 May 2022].

CodeSource.io. 2020. *GraphQL vs SQL - Beginners overview*. [online] Available at: <https://codesource.io/graphql-vs-sql/> [Accessed 25 May 2022].

Baldaniya, H., 2019. *Why and When to Use GraphQL - DZone Integration*. [online] dzone.com. Available at: <https://dzone.com/articles/why-and-when-to-use-graphql-1> [Accessed 25 May 2022].

StackShare. 2020. *Dgraph vs Neo4j* | *What are the differences?*. [online] Available at: <https://stackshare.io/stackups/dgraph-vs-neo4j> [Accessed 4 June 2022].

Neo4j Graph Data Platform. 2021. *Importing CSV Data into Neo4j - Developer Guides*. [online] Available at: <https://neo4j.com/developer/guide-import-csv/> [Accessed 1 June 2022].

Informació Addicional