EPS Escola Politècnica Superior
UdG

Universitat de Girona

GRETA: disseny i desenvolupament d'un chatbot d'assistència virtual

Treball Fi de Grau

Grau en Enginyeria Informàtica. Pla 2012

Document: Memòria
Alumne: Ferran Capallera Guirado
Director/Tutor: Ignacio Martín Campos
Convocatòria: 06/2020

Índex

Capítol 1

Introducció

Motivacions

La idea de dissenyar una aplicació de chat va ser proposada per l'empresa en la que treballo (Vitrosep, S.L.) ja que en aquest últim any estem fent especial esforç en el tema de la digitalització. El voler fer sempre un pas endavant i estar a l'avantguarda comporta experimentar amb tecnologies que encara no estan del tot incorporades al nostre dia a dia i intentar fer-les nostres.

La idea em va semblar genial, ja que no tenia experiència amb cap tecnologia semblant i desitjava veure quins eren els límits i les prestacions d'una aplicació d'aquest tipus, era una bona oportunitat per aprendre una tecnologia nova i poder crear un projecte des de zero pensat per posar en pràctica quan estigués acabat.

D'altra banda una de les meves motivacions també és poder veure quines ampliacions es poder dur a terme al projecte a mesura que vaig veient totes les prestacions del framework, així com incorporar a l'aplicació funcionalitats per ajudar a altres necessitats de l'empresa.

Propòsit

El propòsit d'aquest projecte és dissenyar un chatbot que permeti facilitar la relació amb els clients i estalviar feina automatitzable i repetitiva als treballadors.

Per fer això s'hauran de trobar eines per fer el chatbot i els serveis que el comprendran. Així com també estudiar la usabilitat d'altres chatbots per veure com haurien de ser els fluxs de conversació i la informació que s'ha de guardar de cada conversa per poder fer l'experiència més fàcil per l'usuari.

Una de les principals idees és també mantenir totes les bases de coneixement abstretes de l'aplicatiu principal, perquè es puguin modificar diàlegs i informació de productes i ofertes sense haver de recompilar el codi. Una manera de realitzar això serà treure informació de productes i clients directament del

software de gestió de l'empresa (SAP), que utilitzem com a font centralitzada de coneixement.

Amb la primera versió alfa d'aquest projecte tindrem una eina per experimentar, per testejar amb clients i per millorar constantment. Veurem quines funcionalitats podrem incorporar en el futur i quines modificar.

Objectius

La idea d'aquest projecte és que es posi en pràctica un cop s'arribi a la primera versió alfa. Per tant l'objectiu principal d'aquest projecte és aprendre totes les eines que ens ofereix l'entorn de desenvolupament del chatbot i aconseguir una bona aplicació amb un bon disseny modular per anar afegint funcionalitats i fer més fàcil el manteniment.

- Estudi de les eines, serveis i frameworks per construir un chatbot.
- Estudi de les tecnologies: estudi d'exemples de chatbots actuals.
- Disseny d'un chatbot que permeti reconèixer l'imput dels usuaris i poder elaborar respostes adequades.
- Dissenyar una funcionalitat per poder comprar a través del chatbot com si es tractés d'una botiga virtual.
- Dissenyar un sistema de permisos per tractar diferent a cada usuari segons el seu nivell de privilegis.
- Dissenyar una funcionalitat que permeti ajudar al client amb els possibles problemes tènics relacionats amb els productes de la nostra empresa.
- Dissenyar un diagrama de classes per la persistència de dades necessàries pel funcionament de l'aplicació.
- Estudiar el flux de conversa principal del chatbot perquè sigui còmode i intuïtiu per l'usuari, així com un sistema d'ajuda per guiar-lo.
- Trobar un sistema per comunicar el chatbot amb el software de gestió SAP per importar informació de productes i clients i exportar comandes.

Capítol 2

Estudi de viabilitat

En aquest capítol es mostren tots els recursos necessaris per tal de dur a terme el desenvolupament del projecte, així com els possibles costos.

2.1 Recursos tecnològics

Representa les eines (hardware i software) necessàries per a tal de desenvolupar i mantenir l'aplicació.

- Recursos desenvolupament aplicació mòbil:
  - Un ordinador amb Visual Studio pel projecte net core.
  - El .net framework sdk 4.7.2 i el netcoreapp 2.1 necessaris per desenvolupar l'aplicació.
  - El Bot Framework Emulator per testejar l'aplicació.

Tot i que la botiga virtual amb Prestashop integrat no forma part del projecte en sí, també la llistaré ja que he hagut d'utilitzar en gran part el PrestashopApi per obtenir i enviar dades.

L'eina que es farà servir per hostejar el ChatBot un cop estigui desenvolupat és el portal de Microsoft Azure, ja que la majoria de serveis i recursos cognitius que utilitza la app són part d'azure o microsoft. Amb una suscripció al portal utilitzant diversos serveis ens permet gestionar-ho tot des de la mateixa eina, així que ho he preferit a tenir serveis descentralitzats.

- Recursos utilitzats per l'aplicació mòbil:
  - Una base de dades hostejada a Azure per la persistència de dades.
  - Una botiga virtual online amb prestashop integrat, amb el webservice activat.
  - L'SQL Server Management Studio per dissenyar la base de dades.
  - L'eina ShopSyncro per vincular el software de gestió SAP amb Prestashop de la botiga virtual.

2.2 Costos

Els costos del projecte estaran conformats pel preu i manteniment dels recursos necessaris així com del personal necessari per al desenvolupament i manteniment de l'aplicació.

A la taula [2.1] podem veure els costos dels recursos necessaris.

Recurs	Cost (€)
Ordinador per desenvolupar l'aplicació	650.00
Mòbil amb Telegram	130.00
Total	780.00

Taula 2.1: Costos dels recursos.

Per altra banda, els costos del personal es veuen reflectits a la taula [2.2]. Les tasques estan pensades per dur-se a terme per diferents perfils de personal, però s'ha establert el preu de **6,75 €/hora** com una referència per calcular el preu del personal .

Tasca	Perfil	Hores	Cost (€)
Anàlisi i Disseny de l'aplicació	Dissenyador	100	675.00
Anàlisi i Disseny de la Base de Dades	Programador	16	108.00
Implementació de l'aplicació	Programador	840	5670.00
Documentació	Programador	48	324.00
Total	–	1004	6777

Taula 2.2: Costos del personal.

Aquest cost ha sigut aproximat a la feina que he desenvolupat jo. Tinguent en compte que no coneixia res de la tecnologia a l'hora de començar el projecte. Si s'hagués tractat de professionals del framework, sortirien menys hores però el preu per hora seria més car.

També cal sumar els preus del hosting de serveis, com la base de dades i el servei del Bot (que realment només s'hauria de comptar un cop implantat, ja que durant el desenvolupament no s'ha pagat res):

Programari	Cost (€/mes)
SQL database	92.12
Azure App service	55.15
Total	147.27

Taula 2.3: Costos del programari.

S'ha establert que la despesa en manteniment de aquest aplicatiu rondaria els 10.00 al mes,

En total l'aplicació hauria tingut un cost de **7557 €** més una despesa de **147.27 €** al mes. Cal recordar, però, que no s'ha comptat el manteniment de la zona on treballa l'usuari (internet, llum, etc.) que això faria augmentar el preu

Capítol 3

Metodologia

Per desenvolupar aquest projecte s'ha optat per utilitzar una metodologia semblant al Desenvolupament iteratiu i incremental pels motius que exposaré a continuació:

3.1 Què és el desenvolupament iteratiu?

El desenvolupament iteratiu és una metodologia de desenvolupament que s'executa iterativament, construint l'aplicació en cicles amb el principal avantatge de poder aplicar als cicles següents el que s'ha après als cicles anteriors.

Els cicle inicial consisteix en definir els subproblemes de l'aplicació i construir un prototip d'aquests, i els següents cicles consistiran en, iterativament, millorar el que s'ha construït. Això pot portar com a conseqüència canvis al disseny o modificacions a algoritmes principals agregant idees o mètodes que s'han après a cicles anteriors.

3.1.1 Avantatges

- És útil si saps des de bon principi els requeriments principals de l'aplicació però no tens clara l'implementació.
- Permet l'aprenentatge de l'entorn a mesura que es desenvolupa l'aplicació.
- Permet incorporar noves funcionalitats o refinar les que ja estaven definides utilitzant conceptes rencentment apresos.
- És compatible amb l'aproximació *"prova i error"* que s'ha utilitzat per diversos apartats de l'aplicació.

3.1.2 Inconvenients

- Obliga a revisitar diverses vegades la mateixa funcionalitat o bloc de codi.

- Pot comportar canvis de disseny o reestructuracions de l'aplicació, inclús pot portar a refer completament una funcionalitat.
- És més tediosa i comporta més temps que altres metodologies de disseny.

3.2 Per què s'ha triat aquesta metodologia?

Abans de començar el projecte, l'únic que coneixia eren els requeriments de l'aplicació. No tenia cap idea de quin llenguatge fer servir, quines llibreries ni quin framework. Així que quan vaig decidir quin framework escollir, tot era nou i per aprendre. Tenia clar que el disseny inicial no seria de cap manera el disseny final, i la metodologia de fer funcionalitats senceres d'una en una no era possible, ja que no tenia els coneixements suficients de l'entorn; havia d'aprendre.

Tampoc podia aplicar la metodologia al peu de la lletra, com s'aplicaria en un equip porfessional de desenvolupament de software, ja que això implica estimar riscos i el treball que comportaran els cicles d'iteracions. Jo, en canvi, tot i que coneixia els riscos no sabia quantes hores dedicaria a cada iteració o a cada subproblema de l'aplicació. Per això em refereixo a que he aplicat una metodologia **semblant**. Tot i així considero que ha sigut la millor aproximació al problema.

Per tant, les fases que explicaré a continuació no són exactament les fases del mètode iteratiu i incremental sino les que he aplicat jo per desenvolupar el meu projecte.

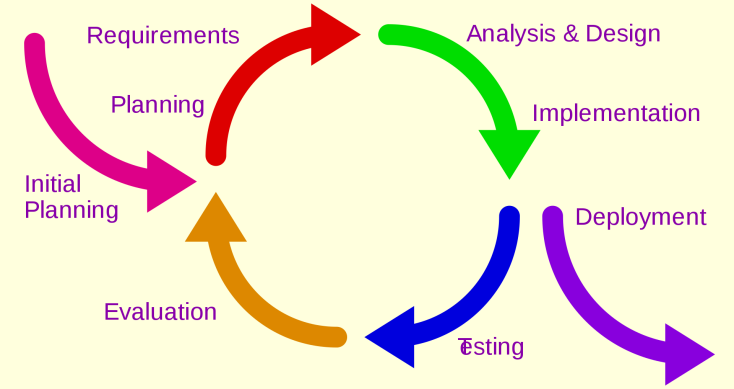


Figura 3.1: Fases de la metodologia iterativa incremental

3.3 Fases del desenvolupament iteratiu i incremental

El desenvolupament iteratiu i incremental està format per dues subfases: una d'inicial, on es farà el plantejament principal de l'aplicació, i una altra subfase en forma de cicle que es repetirà iterativament.

1. **Plantejament principal:** En la fase de plantejament principal es defineix l'abast del projecte: requeriments funcionals i no funcionals.

2. La segona subfase en forma de cicle, que s'executarà iterativament, consta de les següents parts:

   (a) Elaboració d'una solució a alt nivell que compleixi els requeriments no funcionals i elimini els principals problemes de la funcionalitat que s'està tractant en qüestió.

   (b) S'implementa una solució d'acord amb el disseny que s'ha elaborat al pas anterior, comprovant que efectivament s'han complert els requeriments. Si hi ha dubtes a l'hora d'implementar o no es tenen els coneixements necessaris, en aquesta fase es farà recerca de recursos o informació sobre les llibreries que he utilitzat, determinant quina és la millor opció a nivell d'implementació.

   (c) Una fase de testing on es comprovarà que la solució implementada sigui vàlida, obtinguent així un prototip del que seria la solució final del subproblema. En cas d'obtenir resultats inesperats es debugarà i s'adreçarà el problema elaborant una correcció.

   (d) Evaluació del que es pot millorar de la solució implementada o de quins refinaments cal fer. Si la solució encara està incompleta, també es treballarà en el que falta per incorporar aplicant el coneixement que s'ha obtingut en el cicle actual.

Aquesta aproximació es podria aplicar adreçant els subproblemes d'un en un, però la idea és aplicar-lo en tots alhora, cada vegada obtenint una versió completa de l'aplicació més i més refinada.

Capítol 4

Planificació

En aquest capítol parlarem sobre com s'ha planificat el treball i els objectius que s'han anat marcant al llarg del projecte.

Aquest projecte va començar a mitjans de Juliol de 2019 i s'han invertit al voltant de 20-25h per setmana principalment en horari laboral. Els primers mesos es va treballar menys intensament degut a que hi havia un projecte més prioritari a l'empresa però cap a principis d'agost el ritme de treball va augmentar considerablement.

Els objectius es van definir vagament a l'inici del projecte, i un cop em vaig familiaritzar amb les eines escollides per treballar es van definir uns objectius més concisos a nivell funcional i d'aplicació.

Degut a la completa desconeixença del cap d'aplicacions de chat la major part de la feina de les primeres setmanes va ser triar una tecnologia i familiaritzar-se amb aquesta, treball explicat a l'apartat **7. Estudis i decisions**.

- Recerca i familiarització de la millor tecnologia per desenvolupar l'aplicació i definir l'estructura inicial. En la taula [4.1] podem veure les tasques que han estat necessàries.

Tasca	Hores
Recerca i decisió de la tecnologia utilitzada	16
Estudi del funcionament a nivell superficial del framework	12
Estudi dels samples de Microsoft d'implementacions del framework	24
Disseny de l'estructura principal del ChatBot (dispatch)	8
Total	60

Taula 4.1: Tasques disseny de l'estructura base

- Abans de començar a programar i implementar el sistema Dispatch cal tenir llest el NLU, i la base de coneixement del bot. En la taula [4.2] podem veure les tasques que han estat necessàries.

Tasca	Hores
Recerca i decisió de la tecnologia utilitzada	12
Creació d'un resource pack al portal d'Azure	4
Creació d'una base de coneixement simple	8
Vinculació del bot amb la base de coneixement	3
Total	27

Taula 4.2: Tasques del disseny d'una base de coneixement

- Ara que ja hem creat una base de coneixement i l'hem vinculat amb el bot (el bot la pot utilitzar), hem d'aconseguir que utilitzi aquesta base de coneixement en una conversa. En la taula [4.3] podem veure les tasques que han estat necessàries.

Tasca	Hores
Implementació d'un sistema dispatch per utilitzar la base de coneixement	8
Creació de deserialitzadors per obtenir l'informació	6
Implementació del diàleg principal de l'aplicació	12
Total	26

Taula 4.3: Tasques de l'implementació del motor principal del bot

- Ara el bot té l'eina dispatch incorporada. Anem a agregar una capa de funcionalitat bàsica seguint la metodologia incremental. Primer dissenyar la funcionalitat base per comprar productes. En la taula [4.4] podem veure les tasques que han estat necessàries.

Tasca	Hores
Implementació d'una eina bàsica de cerca de productes	4
Implementació d'una eina per mostrar en pantalla els productes de forma intuïtiva	12
Disseny de les cartes de compra i confirmació de compra	6
Implementació del diàleg de compra	16
Implementació de la persistència dels carros de compra de l'usuari	2
Total	40

Taula 4.4: Tasques de l'implementació del diàleg de compra de productes

- Al mateix temps que desenvolupem aquesta nova funcionalitat creo també la funcionalitat d'assistència tècnica. El sistema està preparat per llegir-se a partir d'un txt, la part de la base de coneixement és a càrreg d'un altre departament de l'empresa. En la taula [4.5] podem veure les tasques que han estat necessàries.

Tasca	Hores
Disseny i implementació d'arbre de decisions	8
Persistència de dades d'una sol·licitud d'assistència tècnica	6
Persistència de sol·licitud no exitosa d'assistència tènica	2
Disseny i implementació de parsing d'arbre de decisions	16
Total	32

Taula 4.5: Tasques de l'implementació del sistema d'assistència tècnica

- Una de les altres funcionalitats bàsiques a desenvolupar (que després de certs canvis quedaria obsoleta, una de les conseqüències del desenvolupament iteratiu creixent) és la d'afegir productes. En la taula [4.6] podem veure les tasques que han estat necessàries.

Tasca	Hores
Disseny de l'adaptive card	3
Disseny i implementació del diàleg	4
Implementació de les operacions a la base de dades	1
Total	8

Taula 4.6: Tasques del diàleg per afegir productes

- Un dels objectius principals del projecte era dissenyar un sistema de permisos i privilegis perquè cada usuari pugui interactuar amb les funcionalitats necessàries segons el seu nivell de privilegi (client, usuari no registrat, representant, treballador, etc.). En la taula [4.7] podem veure les tasques que han estat necessàries.

Tasca	Hores
Disseny del sistema de permisos	4
Implementació d'una superclasse amb funcions per controlar els privilegis	8
Persistència dels permisos dels usuaris	2
Implementació del bloqueig de permisos a totes les funcionalitats existents	12
Total	26

Taula 4.7: Tasques del sistema de permisos

- Fins ara tota la persistència de dades s'estava fent mitjançant l'storage del bot. Per tenir un sistema més centralitzat i consistent es va decidir posar les dades en una base de dades. El que va comportar integrar un ORM. Aprofitant que s'incorporava un servei nou i s'havia de reestructurar gran part del codi es va decidir a posar un altre nou servei, l'interacció amb el

Prestashop mitjançant una API. En la taula [4.8] podem veure les tasques que han estat necessàries.

Tasca	Hores
Estudi i decisió de quin ORM utilitzar	8
Implementació de l'ORM mitjançant l'estil table first	6
Canvi de storage a ORM a totes les parts del codi	8
Estudi i decisió de quina eina "retrofit" utilitzar	8
Estudi de la API de Prestashop	24
Implementació del servei REFIT a l'aplicació	2
Disseny de les crides i serialitzadors bàsics	4
Canvi dels productes a storage pels de Prestashop	6
Total	66

Taula 4.8: Tasques de l'incorporació de nous serveis

- Cal un sistema per determinar quin nivell de privilegi té cada usuari, s'ha desenvolupat un sistema de validació dels usuaris. En la taula [4.9] podem veure les tasques que han estat necessàries.

Tasca	Hores
Disseny de l'adaptive card de validació d'usuaris	3
Disseny i implementació del diàleg de valdiació	4
Estudi sobre les notificacions proactives	24
Persistència dels ids interns de cada usuari a temps real	6
Implementació d'un sistema de notificacions proactives	12
Implementació del diàleg de sol·licitud de validació	4
Total	53

Taula 4.9: Tasques del disseny de sistema validació d'un usuari

- Finalment cal crear un sistema per loguejar usuaris. Aprofitant que tenim una botiga on-line (Prestashop) on els usuaris s'han hagut de crear un compte aprofitarem aquestes credencials per el login del bot. En la taula [4.10] podem veure les tasques que han estat necessàries.

Tasca	Hores
Estudi dels sistemes criptogràfics de Prestashop	8
Disseny d'un flux de diàleg de benvinguda	6
Implementació del flux de diàleg de benvinguda (login, registre i valdiació)	24
Validació de contrassenyes utilitzant l'API de prestashop	6
Millora del flux de conversa de tots els diàlegs	16
Total	60

Taula 4.10: Tasques de flux de benvinguda, login i altres flux de conversa

Capítol 5

Marc del treball

En aquest capítol, comentarem el marc de treball del projecte.

Primer parlarem sobre els chatbots (apliacions xat, bots...), alguns exemples en àrees actuals i per què ens hauria d'interessar afegir un ChatBot a la nostra empresa. Al final faré un esment als altres col·laboradors del projecte.

5.1 ChatBots: què necessitem saber?

Un chatbot és un tipus de software que rep, processa i contesta missatges fent possible una conversa amb l'usuari.

A diferència de les aplicacions, aquests no necessiten cap descàrrega ni actualització per part de l'usuari ja que funcionen sobre un **canal**. Si considerem el chatbot com una funció back-end, el canal és la seva aplicació front-end, sent qualsevol aplicació de xat. Les més populars del mercat són:

- Facebook Messenger
- Telegram
- Slack
- Skype
- Microsoft Teams

El fet que els chatbots només requereixin una aplicació que probablement el client ja té instal·lada i hi és familiar és un dels punts forts que tenen els chatbots.

Hi ha diversos tipus de ChatBots i es poden classificar de diverses maneres, algunes d'elles podrien ser:

5.1.1 Segons si integren NLU

L'NLU (*Natural Language Understanding*) és un concepte important que anirà apareixent al llarg de la memòria. L'NLU és l'estudi i comprensió del llenguatge natural, qualitat que dota un ChatBot d'anàlisi d'una frase complexa per entendre-la i elaborar una resposta corresponent.

Per exemple la frase: *Necessito un vol de Roma a Amsterdam per demà a les 12:30.*

Si el chatbot tingués una bona base NLU dissenyada extreuria la següent informació:

- **Intent:** sol·licitud de vol.
- **Origen:** Roma.
- **Destinació:** Amsterdam.
- **Data:** demà (per exemple 21/07/2020) 12:30h.

Ha analitzat un missatge i n'ha extret la informació vital. Dissenyar una bona base de NLU té un cost i altres empreses opten per l'extracció d'informació **step-by-step**. És una bona alternativa que ens estalvia l'NLU i guia l'usuari per preguntar l'informació pas per pas. Per exemple:

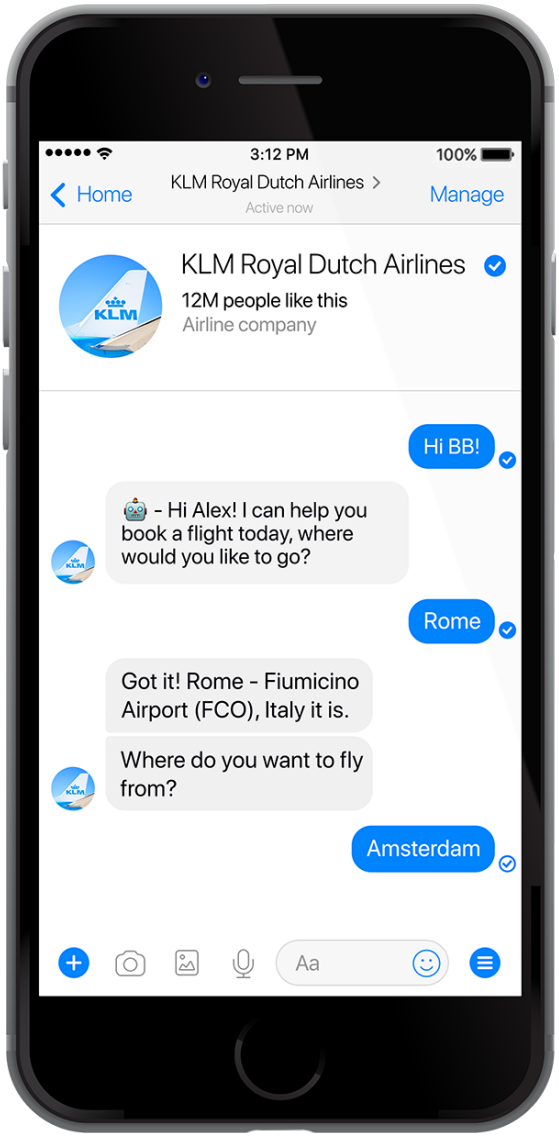


Figura 5.1: Sistema step-by-step

Aquest és un bon exemple d'un cas step-by.step, on no és necessari analitzar una frase sencera ja que el bot guia a l'usuari per extreure l'informació vital d'una sol·licitud de vol aconseguint el mateix objectiu.

5.1.2 Segons el nivell de relació amb l'usuari

A l'hora de dissenyar un bot és molt important estudiar quina informació de l'usuari ens interessa que quedi guardada i en què ens influirà aquesta informació.

Per exemple si dissenyem un bot que serà un guia turístic per un museu, podem considerar un disseny **anònim** que servirà per tots els usuaris del museu. L'usuari es connecta amb el bot, fa preguntes de les zones o obres que va veient i el bot respòn.

En canvi, si en comptes d'un bot guia turístic de museu fos un bot guia turístic de Catalunya, podríem considerar el cas de mantenir certa informació de l'usuari:

- De quina zona és el nostre usuari?
- Quina edat té el nostre usuari?
- Mantenir una llista de les localitats que visita.
- Tenen alguna característica en comú aquestes localitats (Gastronomia, història medieval o modernista, hàbitat rural o urbà, localitat natal de personatges històrics, punt important artístic...)

Aquesta informació ens permetria elaborar una sèrie de recomanacions tinguent en compte els gustos del nostre usuari, establint una relació més forta amb el client i fent del chatbot una eina més potent i imprescindible.

El tipus de chatbot anònim és el que solem trobar en moltes webs, on els usuaris no registrats poden interactuar, demana ajuda tècnica per problemes generals. El processament de les sol·licituds és igual per a tots els clients.

En canvi, chatbots de compra o chatbots on l'interès principal és un fort sistema recomanador cal guardar certa informació de l'usuari i mantindre'l *loguejat* en certa manera. Fet que comportarà considerar altres informacions relatives a la conversa.

- Persistència de les dades essencials de l'usuari.
- Recordar l'estat de la conversa (diàleg).
- Matenir un **token** únic que valida a l'usuari en una conversa.

5.2 Àrees comunes d'aplicació d'un chatbot

Des del 2016 l'ús dels ChatBots i la incorporació a les empreses està incrementant ja que són una eina útil en un gran àmbit d'especialitzacions. Algunes de les funcions més comunes que es deleguen a ChatBots al panorama empresarial actual són:

5.2.1 Atenció al client

El ChatBot no està pensat per substituir completament un treballador d'atenció al client. Nogensmenys, serveix molt bé com a mitjà entre el client i el treballador, si el bot no aconsegueix satisfer les necessitats del client, podrà establir una connexió amb un treballador d'atenció al client.

Molts chatbots tenen preparat una funcionalitat de FAQs o preguntes més freqüents en l'àmbit d'atenció al client.

Un dels altres avantatges és que el ChatBot està disponible 24 hores al dia 7 dies a la setmana, sent capaç de resoldre així problemes que poden ocórrer en horari no laboral.

El chatbot també pot conversar amb il·limitades persones alhora, fent la feina de manera més eficient i concorrent.

5.2.2 Màrqueting

Un chatbot és l'eina perfecte per publicitar productes o funcionar com a canal de vendes. El client és capaç de demanar informació de productes, serveis, etc. Alguns dels bots actuals porten integrades les següents funcions:

- **Canal de vendes:** permet demanar informació de productes i previsualitzar-los dins el canal de chat. També permet fer compres com si es tractés d'una botiga on-line. En alguns casos també pot portar incorporat un sistema de tracking de la comanda, permetent al client saber en quin estat es troba (pendent, enviant) i proporcionant un *TrackId* per visualitzar l'enviament en un mapa.

- **Sistema de subscripcions:** Un client es pot subscriure a un sistema de notificacions que l'avisarà quan certs productes surtin nous al mercat o de noves ofertes que apareixin. Això manté al client pendent i en certa manera en contacte constant amb l'empresa.

5.2.3 Informació

L'era digital ens ha portat a estar consumint informació constantment: ja sigui a les notícies, diaris digitals o xarxes socials. Els chatbots avui en dia són una altra forma de consumir informació.

En l'exemple que veurem a continuació observarem que un usuari pregunta el temps que fa a Madrid. Això és un tipus d'informació que podem obtenir de moltes maneres, la més corrent és comprovar la funció forecast que tenen tots els nostres smartphones.

Però els chatbots tenen un altre avantatge, podem presentar l'informació de la manera que convé a l'usuari. En comptes de preguntar *"Quin temps fa?"* i elaborar ell les seves pròpies conclusions pot preguntar directament *"Necessitaré ulleres de sol?"* i el chatbot, amb una informació bàsica com quin és el temps que fa, pot elaborar una resposta adient per la pregunta de l'usuari.

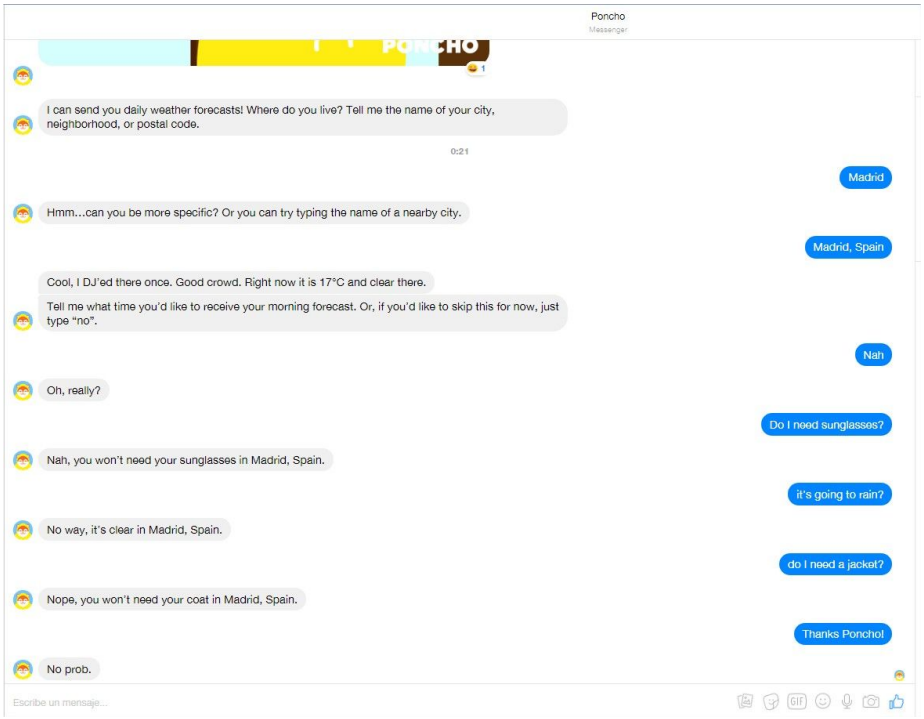


Figura 5.2: ChatBot Poncho, informant del temps d'una forma innovadora

5.3 Per què ens interessa tenir un ChatBot a l'empresa?

Seguidament, discutirem per què ens hauria d'interessar tenir un ChatBot a l'empresa, exposant alguns avantatges i inconvenients d'aquesta tecnologia. Alguns dels avantatges ja s'han explicat anteriorment, en farem un breu resum.

5.3.1 Avantatges

Els avantatges principals de tenir un ChatBot a l'empresa són:

- Servei al client més ràpid i en qualsevol hora.
- Més productius que les sessions de xat en viu.
- Poden donar servei a múltiples clients alhora.
- Ajuden a automatitzar tasques que es fan diverses vegades al dia.
- Són capaços de proporcionar informació personalitzada per un client.
- Generen big data.

- Si el disseny és correcte, la base de coneixement pot ser gestionada per algú que no és expert en el camp de la informàtica.

5.3.2 Inconvenients

Per contra, els desavantatges d'un ChatBot són els següents:

- El chatbot és una eina complexa de desenvolupar, requereix molt de temps, un bon disseny d'aplicació i bona base de coneixement.
- Els chatbots han de ser capaços d'executar accions correctament a part de mantenir converses pregunta-resposta.
- Si el bot no és capaç de guiar a l'usuari, aquest es pot perdre en el flux de la conversa.
- Finalment, el que més m'ha afectat en el meu cas: El ChatBot és una tecnologia emergent. Buscar informació específica és **complicat**, no hi ha formacions i requreix **gran quantitat** de recerca, aprenentatge i disseny per funcionar.

5.4 Altres col·laboradors del projecte

La feina relacionada amb el software ha sigut desenvolupada únicament per mi: recerca, estudi, disseny, implementació. Però hi ha altres aspectes que no farien aquest projecte possible si no hagués sigut per ajuda d'altres col·laboradors.

- El **departament d'enginyeria de Vitrosep** ha ajudat amb la base de coneixement de l'ajuda tècnica, que encara està en construcció. Malauradament, no es pot presentar en aquesta versió del bot però el sistema està funcionant amb una base de coneixement d'exemple.
- L'empresa amb seu a Girona **GSP** (*Global Software Partner*) és l'empresa que ens ha instal·lat el SAP. També ens ofereixen el seu tool framework **SGI Tools** que conté el **SGI ShopSyncro** que permet vincular el SAP amb la botiga virtual Prestashop.
- L'empresa gironina **6TEMS** és la que gestiona i implanta el Prestashop. Ells ens han donat accés a la API webservice de Prestashop, proporcionant-nos una **API Key**. També ens han habilitat un entorn virtual per fer proves amb la seva corresponent API, és l'entorn en el que la demo del meu chatbot funciona, així no afecta el pla productiu de l'empresa.

Capítol 6

Requisits del sistema

En aquest capítol s'exposaran tots els requisits que ha de complir el sistema per tal d'assolir els objectius marcats.

6.1 Requisits no funcionals

Els requisits no funcionals són aquells que indiquen com ha de ser el sistema i no pas que s'ha d'implementar. Aquests són:

- **Seguretat:** El sistema ha de complir amb la normativa de seguretat actual vigent amb referència a la protecció i transferència de dades.
- **Fiabilitat:** El sistema ha de poder funcionar les 24 h al dia sense cap manteniment.
- **Connectivitat:** El sistema ha de poder transmetre dades a través dels diversos serveis als quals està connectat.
- **Escalabilitat:** El sistema s'ha de poder adaptar per tal que el facin servir una gran quantitat de persones.
- **Variabilitat:** El sistema s'ha de poder adaptar a les diferents necessitats dels usuaris. Això comporta que l'usuari sigui capaç d'utilitzar l'aplicació amb fins diferents dels que se li donen en el grup.
- **Usabilitat:** El flux de conversa ha de ser clar i entendor per l'usuari. L'usuari ha d'entendre el que està passant a qualsevol moment de la conversa.

6.2 Requisits funcionals

Els requisits funcionals són aquells que indiquen que ha de fer el sistema. S'han determinat els següents requisits del sistema:

- **Precondicions:** l'usuari disposa d'una aplicació mòbil de xat o la seva corresponent aplicació d'escritori / navegador (Telegram, Facebook Messenger, Slack, Skype...). El bot està deployed a Azure i vinculat amb una identitat (bot) de l'aplicació de xat. O en el cas d'emulació: L'usuari disposa del Bot Framework Emulator i està connectat al bot. El bot està funcionant en una adreça local.
- L'aplicació no té **flux principal** com a tal, sinó que l'usuari requereix al bot la funció que desitja i el bot tria el flux de l'aplicació. Tot això es discutirà al capítol **8. Anàlisi i Disseny del Sistema**.

6.2.1 Requisits del bot

El bot ha de permetre fer aquestes funcionalitats per l'usuari:

- Redirigir a l'usuari a la botiga virtual per **registrar-se**.
- **Loguejar** un usuari utilitzant les credencials de la botiga virtual Prestashop.
- En el cas de superusuaris, **validar** el perfil d'usuaris recentment registrats.
- Posar a disposició de l'usuari la base de coneixement pregunta-resposta per informar-se dels diferents processos de l'empresa.
- Permetre als superusuaris o treballadors de l'empresa **editar fàcilment** la base de coneixement del bot, permetent ampliar l'utilitat del bot.
- Permetre als usuaris visualitzar i comprar els productes de la botiga virtual a través de l'interfície de chat.
- Permetre a l'usuari demanar assistència tècnica per problemes relacionats amb els serveis i maquinària de l'empresa.

Capítol 7

Estudis i decisions

En aquest capítol parlarem sobre els diferents estudis que s'han dut a terme per tal de realitzar aquest projecte així com les decisions que s'han près d'acord amb aquests.

7.1 Recerca i decisió de les tecnologies

En aquesta primera secció explicaré quines tecnologies s'han triat i per què, i a continuació donaré la informació necessària d'aquestes tecnologies per entendre la memòria.

7.1.1 Tecnologia de desenvolupament del bot

Abans de començar el projecte havia de tenir clar quina seria la tecnologia utilitzada. No tenia cap experiència amb aplicacions de chat ni coneixia ningú que n'hagués desenvolupat una. Així que fent recerca a internet vaig trobar dues eines prometedores, les dues tenien un factor en comú: un llenguatge familiar, el C#. Aquestes dues eines eren les següents:

- **BotSharp:** una llibreria prometedora open-source per desenvolupar aplicacions de chat amb machine learning integrat, en aquest cas NLU (*Natural Language Understanding*)
- **Microsoft Bot Framework:** Una extens open-source SDK especialitzat en el desenvolupament d'aplicacions de chat, fet per l'equip de Microsoft.

En primera instància vaig triar la llibreria BotSharp, ja que tenia diversos tutorials a la documentació de com fer servir les eines NLU i em va semblar intuïtiu. Comptava amb una bona GUI, que sempre dona seguretat a nous desenvolupadors com jo que tot just estan començant en el camp de les aplicacions de chat.

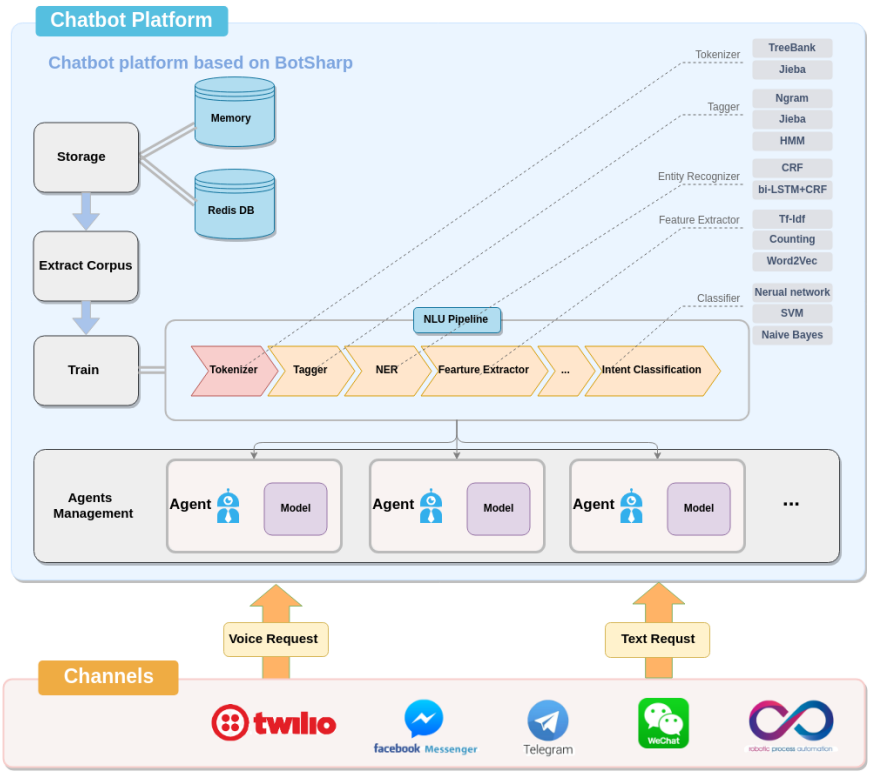


Figura 7.1: Arquitectura BotSharp

També comptava amb la integració de canals (connectar el chatbot amb l'aplicació front-end on el client interactuarà com Telegram, Facebook Messenger...). Aquestes raons em van fer inclinar cap a aquesta eina.

Degut a diversos problemes i complicacions d'instal·lació i després d'haver demanat ajuda a l'apartat d'*Issues* al repositori de GitHub, vaig optar per utilitzar l'altra eina: **Microsoft Bot Framework**.

7.1.2 Tecnologia de connexió a serveis

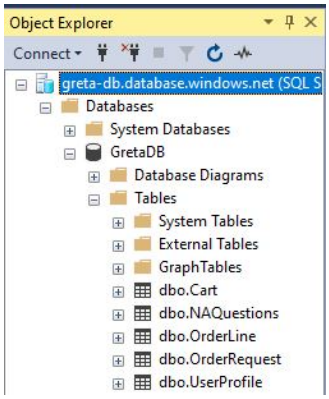
Per connectar amb els diversos serveis externs del bot com la base de dades hostejada a Azure i l'API del webservice de Prestashop, s'han hagut d'utilitzar llibreries adicionals. Principalment un ORM que controla les entities, i una aplicació semblant al *Retrofit*, que ja havíem vist a l'assignatura Projecte de Desenvolupament de Software, per controlar les crides a l'API. Les aplicacions són:

7.1.2.1 Entity Framework Core

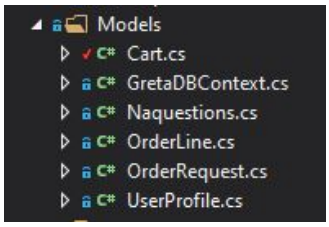
Entity Framework Core és una versió lleugera, extensible, open-source i multiplataforma de la popular tecnologia Entity Framework. Aquesta aplicació m'ha permès comunicar-me de forma fàcil i segura a la base de dades externa, funcionant com a Assignador Relacional d'Objectes (en anglès ORM).

La facilitat d'aquesta tecnologia ve donada per com s'implementa la relació. Hi ha dues formes d'implementar l'assignador:

- **Code First:** l'assignador code first parteix de l'implementació dels Entities. Primer dissenyem les classes amb els seus atributs, i mitjançant un CLI (*Command Line Interface*) que té l'Entity Framework Core ens crea la base de dades a partir del codi dels entities.
- **Table First:** aquesta és l'aproximació que he seguit jo. Primer dissenyem la base de dades. Per fer-ho he utilitzat el **MSSMS** (*Microsoft SQL Server Management Studio*), he definit els objectes amb els seus corresponents atributs i relacions, i mitjançant el CLI de l'Entity Framework m'ha creat automàticament l'implementació dels entities.



(a) Entities dissenyades al MSSMS



(b) Entities creades al Visual Studio

Figura 7.2: Creació dels entities a partir de l'aproximació Table First

7.1.2.2 Refit

Refit és una llibreria inspirada en gran part pel ja conegut **Retrofit**.

El que fa és convertir una API en una interfície al nostre codi. Podem assignar mètodes amb paràmetres a crides de la API, de manera que quan cridem el mètode es realitza la crida al webservice automàticament.

Per obtenir els objectes de retorn d'aquests mètodes hem de crear **serialitzadors**. En el cas de l'API de Prestashop el llenguatge és XML, així que s'han hagut de crear serialitzadors d'aquest tipus (un per cada entitat).

```
    public Cart(Customer customer)
        {
            Id = 0;
            DeliveryAddressId = 0;
            AdressInvoiceId = 0;
            CurrencyId = 1;
            Customer = customer;
            GuestId = 0;
            LanguageId = 7;
            ShopGroupId = 1;
            ShopId = 1;
            CarrierId = 0;
            Recyclable = 0;
            Gift = 0;
            MobileTheme = 0;
            AllowSeparatedPackage = 0;
            DateAdd = DateTime.Now;
            DateUpd = DateTime.Now;
        }


        [XmlElement("id")]
        public override int Id { get; set; }

        [XmlElement("id_address_delivery")]
        public int DeliveryAddressId { get; set; }

        [XmlElement("id_address_invoice")]
        public int AdressInvoiceId { get; set; }

        [XmlElement("id_currency")]
        public int CurrencyId { get; set; }

        [XmlIgnore]
        public Customer Customer { get; set; }

        private int? _customerId { get; set; }


        [XmlElement("id_customer")]
        public int CustomerId
        {
            get { return _customerId ?? Customer.Id; }
            set { _customerId = value; }
        }

        [XmlElement("id_guest")]
        public int GuestId { get; set; }

```

```
50          [XmlElement("id_lang")]
51          public int LanguageId { get; set; }
52
53          [XmlElement("id_shop_group")]
54          public int ShopGroupId { get; set; }
55
56          [XmlElement("id_shop")]
57          public int ShopId { get; set; }
58
59          [XmlElement("id_carrier")]
60          public int CarrierId { get; set; }
61
62          [XmlElement("recyclable")]
63          public int Recyclable { get; set; }
64
65          [XmlElement("gift")]
66          public int Gift { get; set; }
67
68          [XmlElement("gift_message")]
69          public string GiftMessage { get; set; }
70
71          [XmlElement("mobile_theme")]
72          public int MobileTheme { get; set; }
73
74          [XmlElement("delivery_option")]
75          public string DeliveryOption { get; set; }
76
77          [XmlElement("secure_key")]
78          public string SecureKey { get; set; }
79
80          [XmlElement("allow_separated_package")]
81          public int AllowSeparatedPackage { get; set; }
82
83          [XmlIgnore]
84          public DateTime DateAdd { get; set; }
85
86          [XmlElement("date_add")]
87          public string DateAddString
88          {
89              get { return this.DateAdd.ToString("yyyy-MM-dd␣HH:mm:ss"); }
90              set { this.DateAdd = DateTime.Parse(value); }
91          }
92
93          [XmlIgnore]
94          public DateTime DateUpd { get; set; }
95
96          [XmlElement("date_upd")]
97          public string DateUpdString
98          {
99              get { return this.DateUpd.ToString("yyyy-MM-dd␣HH:mm:ss"); }
```

```
100             set { this.DateUpd = DateTime.Parse(value); }
101         }
102
103         [XmlElement("associations")]
104         public CartRowCollection Rows { get; set; }
105     }
```

Listing 7.1: Exemple d'un serialitzador

7.1.3 Eines d'integració NLU

Les dues eines utilitzades per NLU són gratuïtes, desenvolupades per Microsoft i amb un fort vincle amb el Bot Framework. Les dues eines tenen finalitats diferents i doten al bot de la capacitat d'entendre els missatges dels usuaris. Aquestes dues eines són:

7.1.3.1 Qna Maker

El QnA maker (*Question and Answer maker*) és una eina simple però molt útil i és la base de coneixement del bot. El que fa és crear parelles pregunta-resposta per contestar dubtes de l'usuari.

Té un UI força util a la seva pàgina i els parells es construeixen de la següent manera:

Figura 7.3: Exemple de parell pregunta-resposta

A l'esquerra tenim la sèrie de preguntes que portaran a la resposta de la dreta. A l'hora de dissenyar un parell de pregunta-resposta és convenient pensar de quina manera els clients preguntaran ja que quan més maneres de preguntar tinguem, més fàcil serà que quan haguem fet el *training* del QnA, el bot sigui capaç de donar respostes de manera més consistent.

Un cop estem satisfets amb la base de coneixement que hem elaborat, farem el *training* (és el que parlem a les assignatures d'Intel·ligència Artificial quan fem un training de model de dades), i publiquem la base de coneixement. Això es convertirà en una API des d'on el Bot podrà enviar la pregunta de l'usuari per obtenir la resposta de la base de coneixement.

La resposta ve donana amb un JSON, acompanyada de certa informació útlil com:

- La resposta més probable que correspòn a la pregunta.

- L'accuracy score. Quan més s'acosta la pregunta a una de les preguntes que hem afegit a la base de coneixement, més alt serà l'accuracy score.

7.1.3.2 LUIS

LUIS (Language Understanding) és una altra eina de Microsoft que ens ajudarà amb l'intel·ligència del bot.

A diferència del QnA, el LUIS no només reconeix una frase per donar una resposta sino que **l'analitza**. De la frase n'extreurà les següents dades:

- **Intent:** l'Intent d'una frase és, per dir-ho d'alguna manera, la seva finalitat. Per exemple la frase *I want to buy a product* té com a intent ComprarProductes, i la frase *How do I log in?* té l'intenció de loguejar-se. En el meu esquema d'aplicació els intents correspondran a funcionalitats.
- **Entities:** els entities seran les peces d'informació que contindran les frases dels usuaris. Agafant un exemple anterior:

  I want to catch a flight from Rome to Amsterdam tomorrow at 12:30 N'obtindrem la següent informació:

  - **Intent:** BookFlight.
  - **Entities:** Rome, Amsterdam, tomorrow, 12:30.

  Podríem tenir dues entities *Ciutat*, una entity *Data* i una entity *Hora*. Inclús dintre l'entity *Ciutat* podríem definir dos **rols**: Origen i Destí. I dir-li al LUIS que després del from ve l'origen i després del to ve el destí.

Veurem el panell de gestió a continuació:

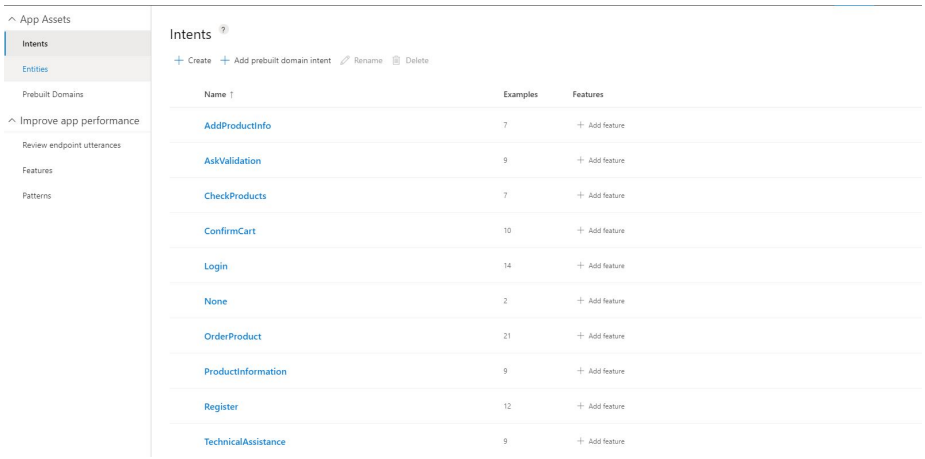


Figura 7.4: Panell de gestió de intents

Com veiem en aquest panell de gestió, tenim tots els intents amb les seus corresponents exemples dintre. Per entendre-ho millor, anem a veure què hi ha dintre del intent *OrderProduct*:

Figura 7.5: Frases dintre l'intent OrderProduct

La idea és pensar de totes les maneres que un client intentarà comprar un producte utilitzant el chatbot (aquestes només són algunes de les que hi ha dins). Com veiem hi ha certes paraules que estan marcades, aquests seran els nostres entities dins la frase.

Hi ha entities que estan predefinits com els nombres o les dimensions, però també es poden definir entities noves, com he fet jo amb els Productes. Al LUIS hi ha diversos tipus d'entities però jo principalment en faig servir dues:

- **Simple:** els simple entities és simplement text. Li indiques al LUIS que en certa posició de la frase hi haurà una ciutat, i a l'hora d'aplicar el reconeixement et reconeixerà el que hi hagi en aquella posició de la frase com a Ciutat. Això dona molta llibertat però també pot donar error. Més correcte seria tenir una llista de totes les ciutats per evitar reconèixer una frase errònia, el que ens porta al següent tipus d'entity.
- **List:** el que ens permet el List entity, a diferència de l'anterior, és definir tots els valors que podrà prendre aquell entity. Com per exemple he fet jo amb els productes:

Figura 7.6: Producte com a List entity al meu panell

Com veiem, els valors normalitzats estan a l'esquerra, i a la dreta hi ha la llista de sinònims. Per exemple si un client demana *zip ties*, el LUIS ho reconeixerà com a nom de producte normalitzat *plastic tie*. I si el producte que ha demanat un client no existeix, el valor de l'entity serà buit.

7.2 Bot Framework: un resum del funcionament

Com ja hem dit, el bot és un software on els usuaris interactiuen d'una forma semblant a una conversa. Cada interacció entre el bot i l'usuari (o viceversa) l'anomenem **Activity**.

Els missatges s'envien entre el servei del bot, i l'aplicació on el bot està connectat: el **canal** (Telegram, Skype, Slack, etc.). Aquest és un exemple d'un echo bot, és a dir, un bot que simplement retorna el missatge que li has enviat.

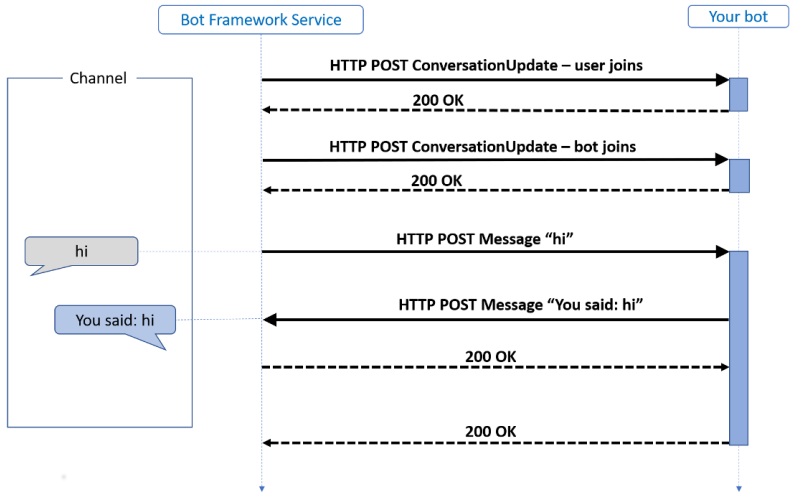


Figura 7.7: Esquema d'activities d'un echo bot

Cada vegada que el bot rep un missatge ho anomenarem **turn**. Això el que farà és cridar la funció de la classe Bot que gestionarà el missatge, mètode que s'anomena **OnTurn**.

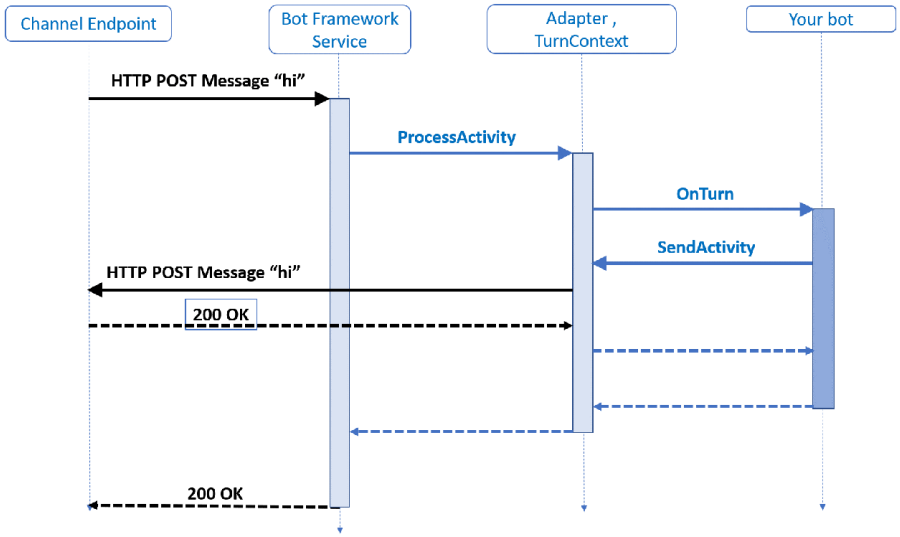


Figura 7.8: Esquema de processament d'activitats

Aquest mètode **OnTurn** és obligatori implementar-lo ja que serà el motor principal del nostre bot. Des d'aquí podrem cridar **Diàlegs** (que veurem a

continuació), enviar missatges, etc.

Una peça molt important de la gestió del torn és el **Middleware** que és tot el que fem abans i/o després de gestionar el missatge. Per exemple, guardar un log de tots els missatges que arriben, o realitzar una sèrie d'accions després d'enviar el missatge. L'esquema d'una gestió de missatge amb middleware seria el següent:

```
1 public override async Task OnTurnAsync(ITurnContext turnContext,
      ↪ CancellationToken cancellationToken)
2       {
3                       //MIDDLEWARE
4
5                       //GESTIO DEL MISSATGE
6
7                       //MIDDLEWARE
8       }
```

Listing 7.2: Gestió del torn amb Middleware

Aquest paràmetre de la capçalera **turnContext** conté la informació de l'Activity:

- El contingut de l'activity (text)
- Qui l'ha enviat?
- Qui l'envia?
- etc.

Com el seu nom indica és el context del torn. Per tots els mètodes on viatja el missatge (més endavant veurem el flux de l'aplicació) es va passant aquest objecte constantment perquè conté la informació vital per executar-se el torn.

7.2.1 Classes Diàleg

Per donar més complexitat a l'interacció amb el bot podem generar Fluxos de diàleg. Una classe Diàleg és un singleton que conté un **DialogSet**, dintre d'aquest DialogSet es poden afegir diferents tipus d'elements però em centraré bàsicament en els que he fet servir jo:

7.2.1.1 Prompts

Els prompts són eines essencials pel funcionament del projecte, ja que permeten demanar a l'usuari informació que serà llegida, aquests són els tipus de prompt que més he utilitzat:

- **TextPrompt:** demana a l'usuari una informació de text, el que escrigui l'usuari com a resposta quedarà enregistrat com a valor del resultat del prompt.

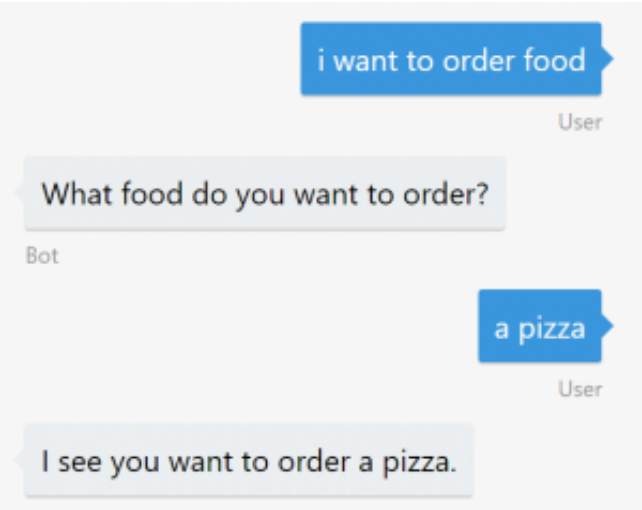


Figura 7.9: Exemple de TextPrompt

- **ChoicePrompt:** demana a l'usuari que trii entre una de les opcions presentades en forma de botó. Aquestes podrien ser Sí/No (**ConfirmationPrompt**) o definir tu les opcions manualment.

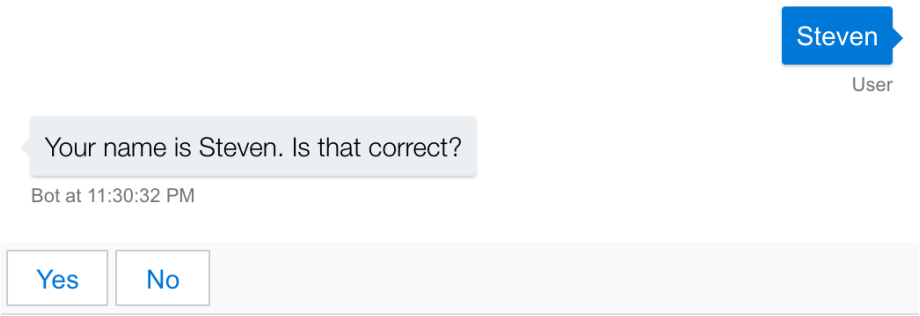


Figura 7.10: Exemple de ConfirmationPrompt

Hi ha una forma molt interessant per validar el valor d'aquests Prompts que es diu **PromptValidator**. A l'hora d'afegir el prompt al DialogSet s'associa una funció que retorna un valor booleà amb el Prompt. Si la validació del prompt fos errònia (el validador retorna fals) el prompt es podria tornar a repetir, o posar fi al diàleg. Això és interessant per comprovar el resultat de la informació enviada per l'usuari.

```
1       private async Task<bool>
            ↪ ValidateEmailAsync(PromptValidatorContext<string>
            ↪ promptContext, CancellationToken cancellationToken)
```

```
2       {
3           var email = promptContext.Context.Activity.Text;
4
5           var userCollection = await
                ↪ PrestashopApi.GetCustomerByEmail(email);
6
7           return await Task.FromResult(userCollection.Elements.Count !=
                ↪ 0);
8       }
```

Listing 7.3: Exemple de PromptValidator dins del meu projecte

Aquest és un exemple dins del diàleg de Login, un PromptValidator associat al TextPrompt que demana el correu electrònic de l'usuari. Si no el troba, li torna a demanar.

7.2.1.2 Diàlegs

Hi ha molts tipus de diàleg dintre el Bot Framework, però per no fer aquesta secció massa llarga em centraré en els tipus de diàleg que he utilitzat en el meu projecte: els diàlegs **WaterFall**.

Els diàlegs Waterfall defineixen una sèrie d'interaccions step-by-step que es duran a terme seqüencialment (WaterfallSteps). Aquí podem veure el DialogSet d'un dels meus diàlegs on, al constructor, defineixo els Prompts que s'utilitzaran i defineixo el Flux de diàleg mitjançant els WaterfallSteps:

```
1           AddDialog(new TextPrompt("EmailValidator",
                ↪ ValidateEmailAsync));
2           AddDialog(new TextPrompt("PasswordValidator",
                ↪ ValidatePasswordAsync));
3           AddDialog(new TextPrompt(nameof(TextPrompt)));
4           AddDialog(new ConfirmPrompt(nameof(ConfirmPrompt)));
5           AddDialog(new WaterfallDialog(nameof(WaterfallDialog), new
                ↪ WaterfallStep[]
6           {
7               AskEmailStepAsync,
8               ConfirmPasswordStepAsync,
9               DisableCardStepAsync,
10              CheckUserProfileStepAsync,
11              AskForVerificationStepAsync
12          }));
```

Listing 7.4: Definició del DialogSet dintre el constructor

Això és dins del constructor del Diàleg de Login. Veiem que afegeixo els diferents prompts que s'utilitzaran amb les seves corresponents funcions de validació i també veiem la definició dels WaterfallSteps. Veiem que el flux de diàleg serà el següent:

- Preguntar l'email

- Preguntar la contrassenya
- Comprovar que el perfil existeix
- Preguntar a l'usuari si vol ser validat (en cas de que sigui un usuari nou)

Per entendre què hi ha dins d'aquestes funcions fem un cop d'ull al primer step, el de preguntar l'email:

```
1       private async Task<DialogTurnResult>
            ↪ AskEmailStepAsync(WaterfallStepContext stepContext,
            ↪ CancellationToken cancellationToken)
2       {
3           var promptOptions = new PromptOptions
4           {
5               Prompt = MessageFactory.Text("Tell␣me,␣what␣e-mail␣do␣you␣
                    ↪ have␣associated␣in␣your␣vitrosepStore␣account?␣" +
6               "Your␣e-mail␣works␣like␣an␣username,␣I␣will␣be␣able␣to␣
                    ↪ find␣you␣in␣your␣database␣instantly!"),
7               RetryPrompt = MessageFactory.Text("Couldn't␣find␣that␣
                    ↪ e-mail␣anywhere,␣make␣sure␣it's␣written␣
                    ↪ correctly.\n" +
8               "So␣again,␣what's␣your␣e-mail?")
9           };
10
11          return await stepContext.PromptAsync("EmailValidator",
                ↪ promptOptions, cancellationToken);
12      }
```

Listing 7.5: Contingut del step de preguntar e-mail

Aquest és exemple d'un step molt simple. Veiem que al final del mètode hi ha una crida a un Prompt. Això és per què la majoria de WaterfallSteps acaben en una pregunta a l'usuari abans de passar el següent step. I un cop l'usuari respòn, s'obté el valor de la resposta al següent step mitjançant l'objecte de la capçalera `WaterfallStepContext`, que conté informació semblant a l'abans esmentat `TurnContext` més informació adicional per saber en quin step està la conversa.

No tots els steps han d'acabar en un prompt, també pot ser un step on no s'interactuï amb l'usuari i només es facin gestions internes de l'aplicació (Com el `DisableCardStepAsync` del codi 7.4, que explicaré més endavant de què serveix).

7.2.1.3 DialogStack

Un concepte molt important d'entendre és el **DialogStack**. El DialogStack és una pila que controla quin diàleg està executant actualment l'usuari. Per interactuar amb el DialogStack tenim diversos mètodes:

- **BeginDialogAsync:** quan cridem aquesta funció (passant per paràmetre el nom del diàleg) afegeix el diàleg sobre el DialogStack. Si l'usuari ja estava en un diàleg, aquest es posarà en pausa fins que el nou diàleg de la pila acabi. Llavors continuarà l'anterior.
- **EndDialogAsync:** acaba el diàleg sobre la pila DialogStack i resumeix el de sota (si n'hi havia algun)
- **ReplaceDialogAsync:** elimina el diàleg de sobre la pila i n'afageix un de nou (el que s'ha passat per paràmetre).
- **EndAllDialogsAsync:** esborra la pila de diàlegs.

Aquesta és la manera amb la que es controla el flux de conversa entre diàleg i diàleg. Si s'utilitza el DialogStack correctament es poden generar diàlegs complexos que redirigeixen a l'usuari segons les seves necessitats. És molt important fer saber a l'usuari que sempre pot tornar enrere o iniciar el diàleg d'una altra manera.

7.3 Adaptive Cards

Els AdaptiveCards són una eina interactiva per presentar la informació de forma esquemàtica i agradable, que he utilitzat al llarg del projecte.

Es poden dissenyar utilitzant JSON o amb codi, mitjançant SDK. S'envien a l'usuari mitjançant una Activity, en comptes de contenir text contindrà un JSON amb la informació de l'adaptive card.

Un dels avantatges dels AdaptiveCards és que, com el seu nom indica, són Adaptive. Els usuaris construeixen el JSON, i el canal (Telegram, Skype, Slack...) s'encarrega de la renderització. Cada canal pot fer una renderització en pantalla diferent, adequat a l'estil d'aplicació. Per això si veiem una tarjeta (AdaptiveCard) del Telegram o una d'Skype seran possiblement diferents.

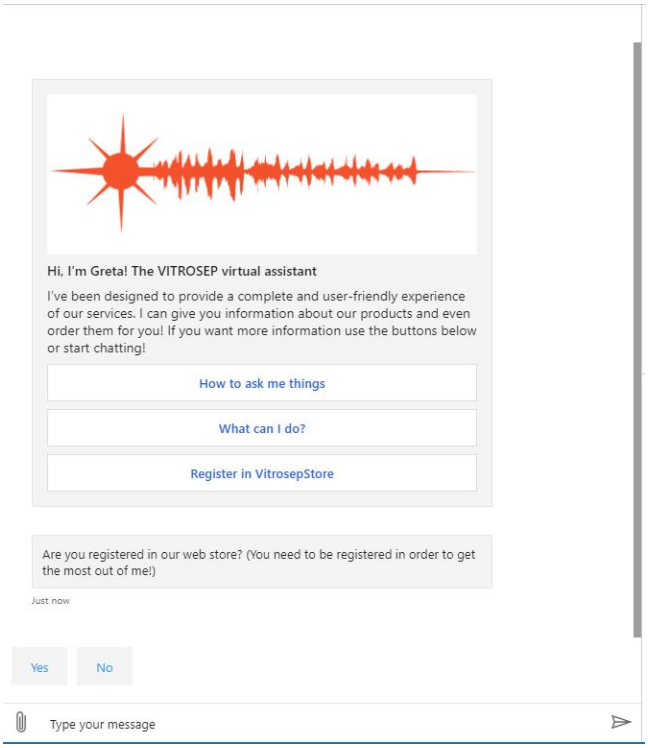


Figura 7.11: Exemple d'AdaptiveCard utilitzada en la benviguda del bot. A sota, un ConfirmPrompt

Com veiem aquest és un AdaptiveCard que he dissenyat pel diàleg de benvinguda del bot. Conté elements multimèdia, text, i també conté botons interactius.

Aquests botons es diuen Actions i poden ser de diversos tipus, els que més he utilitzat a l'aplicació són els següents:

- **Submit:** al clicar el botó envia un Activity cap el bot, un dels camps conté una informació en forma de JSON. El bot ha d'estar preparat per interceptar aquesta informació, deserialitzar-la i elaborar una resposta adient.
- **OpenURL**: el botó que veiem a la figura de dalt, obre una URL directa a la pàgina de registre de l'empresa.

Un dels problemes principals d'aquests botons d'acció és que l'usuari els pot prémer en qualsevol moment de la conversa, encara que siguin antics. El bot pot no estar preparat per rebre una resposta d'aquest tipus i alterar completament el flux de la conversa. La solució d'aquest problema serà adreçada més endavant.

7.4 Dependency Injection, Startup i appsettings

Els Diàlegs (la major part del funcionament del bot) són classes Singleton. Això vol dir que per tot el bot hi haurà una instància única de cada diàleg.

Els recursos i serveis que utilitzen els diàlegs s'hauran d'injectar al constructor, ja que aquests s'instancien únicament al principi de l'execució de l'aplicació. Això vol dir que tots els recursos que necessitem utilitzar dins dels diàlegs s'hauran d'injectar.

Tots aquests elements es defineixen a la classe **Startup**, igual que les configuracions dels serveis que utilitza el bot. Aquesta és la meva classe de configuració:

```
1  public void ConfigureServices(IServiceCollection services)
2  {
3      services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
4
5      // Create the Bot Framework Adapter with error handling enabled.
6      services.AddSingleton<IBotFrameworkHttpAdapter,
           ↪ AdapterWithErrorHandler>();
7
8      // Create the storage we'll be using for User and Conversation
           ↪ state. (Memory is great for testing purposes.)
9      services.AddSingleton<IStorage, MemoryStorage>();
10
11     // Create the User state. (Used in this bot's Dialog implementation.)
12     services.AddSingleton<UserState>();
13
14     //Afegim el servei de QnA i LUIS
15     services.AddSingleton<IBotServices, BotServices>();
16
17     // Creem un ConversationState que ens ajudar[U+FFFD] amb els nodes
           ↪ de conversa del TechSupport.
18     services.AddSingleton<ConversationState>();
19
20     services.AddSingleton<TechnicalAssistanceDialog>();
21
22     services.AddSingleton<AskUserInfoDialog>();
23
24     services.AddSingleton<OrderProductDialog>();
25
26     services.AddSingleton<ConfirmOrderDialog>();
27
28     services.AddSingleton<AddProductInfoDialog>();
29
30     services.AddSingleton<UserValidationDialog>();
31
32     services.AddSingleton<UserLoginDialog>();
33
34     services.AddSingleton<CartToOrderDialog>();
35
```

```
36      // The Dialog that will be run by the bot.
37      services.AddSingleton<MainDialog>();
38
39      services.AddSingleton<NotifyController>();
40
41      services.AddSingleton<UserController>();
42
43      services.AddSingleton<QuestionController>();
44
45      services.AddSingleton<PurchaseController>();
46
47      services.AddSingleton<PermissionDialog>();
48
49      services.AddSingleton<CartToOrderDialog>();
50
51      // Create the bot as a transient. In this case the ASP Controller is
            ↪ expecting an IBot.
52      services.AddTransient<IBot, GretaBot<MainDialog>>();
53
54      services.AddSingleton<ConcurrentDictionary<string,
            ↪ ConversationReference>>();
55
56      var apiKey = Configuration.GetSection("PrestashopSettings")
57          .GetSection("ApiKey").Value;
58      var storeUrl = Configuration.GetSection("PrestashopSettings")
59          .GetSection("StoreUrl").Value;
60
61      String encoded = Convert.ToBase64String(System.Text.Encoding
62          .GetEncoding("ISO-8859-1").GetBytes(apiKey));
63
64      // Afegim la API i li donem una configuracio.
65      services.AddRefitClient<IPrestashopApi>(
66          new RefitSettings
67          {
68              ContentSerializer = new XmlContentSerializer()
69          })
70          .ConfigureHttpClient(c => new HttpClient(new
                ↪ UriQueryUnescapingHandler()))
71          .ConfigureHttpClient(c => c.BaseAddress = new Uri(storeUrl))
72          .ConfigureHttpClient(c =>
                ↪ c.DefaultRequestHeaders.Add("Authorization", "Basic␣" +
                ↪ encoded));
73
74      services.AddDbContext<GretaDBContext>(options =>
75          options.UseSqlServer(
                ↪ Configuration.GetConnectionString("DefaultConnection" )));
76  }
```

Listing 7.6: Startup: Configuració de l'aplicació

Tota la informació sensible que s'utilitza dins d'aquesta classe es troba dins d'un arxiu JSON que es diu `appsettings.json`. Malauradament, aquest arxiu no està al repositori degut a que no es pot fer públic. Però és un arxiu essencial per fer funcionar una aplicació (si desitjeu executar el bot em podeu demanar l'arxiu al correu fcapallera@gmail.com).

Capítol 8

Anàlisi i disseny del sistema

8.1 Disseny del sistema

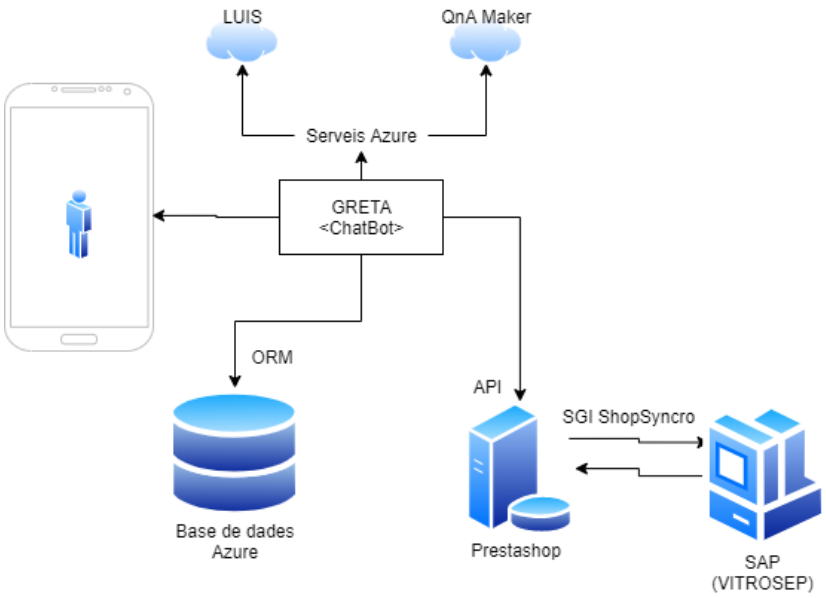


Figura 8.1: Diagrama de comunicacions del sistema

8.2 Anàlisi del sistema

En el cas particular del meu projecte, principalment no tractarem de casos d'ús sino de fluxos de conversa. Els fluxos de conversa és el que dona vida al bot i la manera que té l'usuari d'utilitzar l'aplicació com a tal.

El bot s'ha dissenyat segons un sistema Dispatch.

8.2.1 Què és un sistema Dispatch

El sistema Dispatch és una implementació del bot que permet decidir d'entre tots els fluxos de diàleg i serveis que pot utilitzar, quin és l'adient segons l'input de l'usuari.

El sistema Dispatch està organitzat de la següent manera:

- Una classe **Bot** que incia el procés de conversa cada vegada que rep un missatge. (Recordem el mètode OnTurn)
- Un diàleg principal **MainDialog** que està lligat directament al bot. Aquest és el diàleg que el Bot crida cada vegada que rep un missatge i és el motor principal del bot. El MainDialog és un `WaterfallDialog` que conté els següents steps:

  1. El primer step llegeix el missatge de l'usuari i decideix si s'ha de contestar mitjançant el servei QnA (pregunta-reposta) o ha sigut reconegut com a un Intent del LUIS. En cas de que sigui pregunta-resposta contestarà automàticament i acabarà el flux de diàleg. En cas contrari anirà al següent step.
  2. El segon step és el cas que el missatge de l'usuari ha sigut classificat en un Intent del LUIS. Aquest step el que farà és redirigir a l'usuari al Diàleg corresponent del seu Intent, afegint el nou diàleg al DialogStack. Recordem que al fer això, ara l'usuari comença un nou flux de conversa, i quan s'acabi aquest nou Diàleg tornarà altre cop al MainDialog.
  3. L'últim step és simplement per acabar el flux de conversa amb un *What else can I do for you today?*

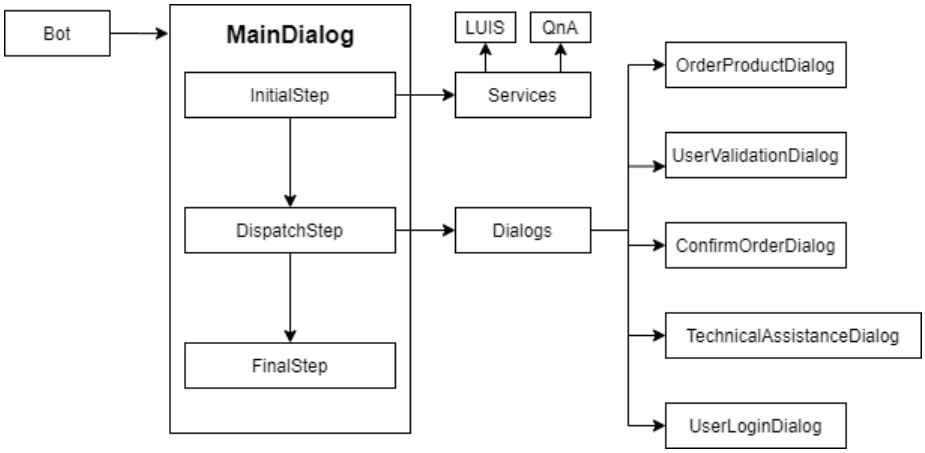


Figura 8.2: Esquema Dispatch del Bot

Cal afegir que el MainDialog té un pas extra a l'inici que no surt a l'esquema. La primera interacció que té l'usuari amb el bot el llença a un Diàleg de benvinguda que només s'executarà un cop. Aquest Diàleg li dona informació sobre el bot i el redirigeix automàticament al diàleg de Login, on es podrà loguejar i començar a fer servir els serveis del Bot.

8.2.2 Fluxos de conversa

En aquest apartat parlarem dels fluxos de conversa de les diverses funcionalitats, com si es tractessin de diagrames de casos d'ús.

El meu cas, el del software de chat, és particular perquè l'activitat entre client i aplicació és intercalada tota l'estona: missatge de client - missatge d'aplicació. Ambdós van seguint el flux de conversa simultàniament i ambdós hi participen. Per això he decidit organitzar les descripcions d'aquesta manera, oposadament a fer diagrames de casos d'ús.

Diàleg de Benvinguda

A continuació descriuré el flux de diàleg Aplicació - Usuari de la funcionalitat de benvinguda.

- **Precondició:** és la primera interacció entre l'usuari i el bot.
- **Permisos:** Usuari no registrat
- El ChatBot inicia la conversa
- Flux de conversa:
    1. El ChatBot envia un AdaptiveCard de benvinguda al client, amb dos botons d'informació i un que redirigeix a la pàgina de registre de VITROSEP. Els botons d'informació es fer servir en qualsevol moment i no queden inhabilitats.
    2. El ChatBot automàticament envia un `ChoicePrompt` (sí/no) preguntant si l'usuari està registrat.
    3. L'usuari contesta a la pregunta sí/no.
        (a) **Sí:** es redirigeix l'usuari al diàleg de login
        (b) **No:** es redirigeix a la pàgina web de registre VITROSEP, es fa saber a l'usuari que un cop registrat, un treballador de VITROSEP validarà el seu perfil. S'acaba el diàleg.

Diàleg de login d'usuari

A continuació descriuré el flux de diàleg Aplicació - Usuari del login d'un usuari.

- **Precondició:** el bot ha redirigit l'usuari o l'usuari ha demanat el bot per loguejar-se.

- **Permisos:** usuari no registrat.
- El bot o l'usuari poden començar aquesta conversa segons la precondició.
- Flux de conversa:
  1. (Opcional) L'usuari ha demanat al bot que vol loguejar-se.
  2. El bot pregunta a l'usuari el seu correu electrònic, que funciona com a nom d'usuari pel login.
  3. L'usuari proporciona el correu electrònic.
     (a) **Correcte:** si el correu és correcte es continua el flux de conversa.
     (b) **Incorrecte:** si no troba el correu electrònic es torna al pas 4.
  4. El Bot mostra un AdaptiveCard per emplenar la contrassenya associada amb el correu.
  5. L'usuari escriu la seva contrassenya i prem el botó d'enviar.
     (a) **Correcte:** si el correu és correcte es continua el flux de conversa.
     (b) **Incorrecte:** si no troba el correu electrònic es torna al pas 4.
  6. El bot dona a conèixer a l'usuari que el login s'ha executat correctament i s'acaba el diàleg.

Diàleg de validació d'usuari

A continuació descriuré el flux de diàleg Aplicació - Usuari de la funcionalitat de validació d'un usuari.

- **Precondició:** el bot ha enviat una notificació a un superusuari o un superusuari ha demanat al bot que vol registrar un usuari.
- **Permisos:** superusuari.
- El bot o l'usuari poden començar aquesta conversa segons la precondició.
- Flux de conversa:
  1. El pas 1 està alterat per la precondició segons qui hagi iniciat la conversa:
     (a) **Bot:** el bot fa saber mitjançant una notificació que un usuari requereix validació i va directament al pas 5.
     (b) **Superusuari:** el bot pregunta el nom de l'usuari a validar.
  2. L'usuari entra el nom de la persona a validar.
  3. El bot busca la persona mitjançant l'API de Prestashop
     (a) **Resultat únic:** Passa directe al pas 5.
     (b) **Múltiples resultats:** el bot mostra una sèrie d'usuaris que corresponen amb el nom que ha entrat el superusuari.

   (c) **Cap resultat:** el bot dona a conèixer al superusuari que no s'ha trobat cap usuari amb aquest nom i es torna al pas 1b.
4. El superusuari sel·lecciona un dels usuaris de la llista.
5. El bot mostra un ChoicePrompt amb tots els nivells de permisos per assignar a l'usuari.
6. L'usuari sel·lecciona un dels nivells de permisos mostrats.
7. El bot assigna el permís a l'usuari sel·leccionat i el marca com a validat. Seguidament, fa saber al superusuari que el perfil ha quedat validat. També envia una notificació a l'usuari validat fent-li saber que el seu perfil està validat i ja pot loguejar-se. S'acaba el diàleg.

Diàleg de compra de productes

A continuació descriuré el flux de diàleg Aplicació - Usuari de la funcionalitat comprar un producte.

- **Precondició:** l'usuari demana al bot que vol comprar un producte.
- **Permisos:** Representant.
- L'usuari comença la conversa.
- Flux de conversa:
  1. L'usuari demana al bot que vol comprar un producte, opcionalment donant tota la informació dins la mateixa frase (producte, quantitat).
  2. El bot extreu els entities del resultat del LUIS.
     (a) Si no hi ha producte es passa al següent pas.
     (b) Si no hi ha quantitat es passa al pas 8.
     (c) Si les dues informacions estan presents es passa al pas 10.
  3. El bot pregunta a l'usuari quin producte vol comprar.
  4. L'usuari contesta especificant el nom d'un producte
  5. El bot busca el producte mitjançant l'API de prestashop
     (a) **Resultat únic:** Passa directe al pas 7.
     (b) **Múltiples resultats:** el bot mostra una sèrie de productes que corresponen amb el nom que ha entrat l'usuari.
     (c) **Cap resultat:** el bot dona a conèixer al superusuari que no s'ha trobat cap producte amb aquest nom i es torna al pas 3.
  6. L'usuari sel·lecciona un dels productes mostrats pel bot.
  7. Aquest pas depèn de l'extracció de dades del LUIS.
     (a) **Sí** el LUIS ha reconegut la quantitat es passa al pas 10.
     (b) **No** ha reconegut la quantitat, es passa al següent pas.

8. El bot pregunta la quantitat a l'usuari.
9. L'usuari respòn amb una quantitat
10. El bot envia una petició de confirmació de si l'usuari vol afegir una línia de comanda amb el producte i la quantitat especificada.
11. L'usuari respòn a la pregunta
    (a) **Sí:** es passa al següent pas.
    (b) **No:** s'acaba el diàleg.
12. El bot genera una línia de comanda amb el producte, la quantitat i l'usuari i l'envia a la base de dades. Seguidament fa saber a l'usuari que la línia de comanda s'ha afegit correctament.

Diàleg de finalització de compres

A continuació descriuré el flux de diàleg Aplicació - Usuari de la funcionalitat de crear una comanda a partir dels productes que ha demanat.

- **Precondició:** l'usuari demana al bot que vol fer la comanda dels productes que ha demanat.
- **Permisos:** Representant.
- L'usuari comença la conversa.
- Flux de conversa:
    1. L'usuari demana al bot que vol fer la comanda amb els productes.
    2. El bot comprova que l'usuari tingui una comanda activa amb productes dins.
        (a) **Sí:** avança al següent pas:
        (b) **No:** el bot notifica a l'usuari que per confirmar la comanda primer n'ha de fer una i acaba el diàleg.
    3. El bot mostra a l'usuari un AdaptiveCard amb la informació de la comanda amb 3 botons disponibles:
        (a) **Confirmar:** passa al següent pas.
        (b) **Cancel·lar**: passa al següent pas.
        (c) **Continuar comprant:** acaba el diàleg.
    4. El bot envia una última petició de confirmació per saber si l'usuari vol Confirmar/Cancel·lar la comanda.
        (a) **Sí:** El flux de conversa avança al següent pas.
        (b) **No:** Acaba el diàleg.
    5. Aquest pas depèn del botó que s'hagi premut al pas 3:
        (a) **Confirmar:** El bot crea un objecte Order a la base de dades i l'envia a l'API dePrestashop (pendent). Fa saber a l'usuari que la comanda s'ha enviat correctament.

   (b) **Cancel·lar**: Elimina les línies de comanda. Fa saber a l'usuari que s'ha cancel·lat la comanda.

6. Acaba el diàleg.

Diàleg d'assistència tècnica

A continuació descriuré el flux de diàleg Aplicació - Usuari de la funcionalitat de demanar assistència tècnica

- **Precondició:** l'usuari demana al bot assistència tècnica per la seva màquina.
- **Permisos:** Client:
- L'usuari comença la conversa.
- Flux de conversa:

  1. L'usuari demana assistència tècnica per la seva màquina.
  2. El bot, seguint l'arbre de decisions de la base de coneixement, fa una pregunta a l'usuari sobre quin problema té. El tipus de prompt és ChoicePrompt, cada un condueix a un node diferent.
  3. L'usuari tria una de les opcions.
  4. Aquest pas ve condicionat per l'estat del node actual:
     (a) Si s'ha arribat a un node fulla (no hi ha més opcions) el dubte de l'usuari queda registrat a la base de dades i s'envia a un treballador de VITROSEP per mitjà d'una notificació. Acaba el diàleg.
     (b) Si no s'ha arribat al node fulla, el bot se situa al següent node i torna al pas 2.

8.2.3 Diagrama de classes

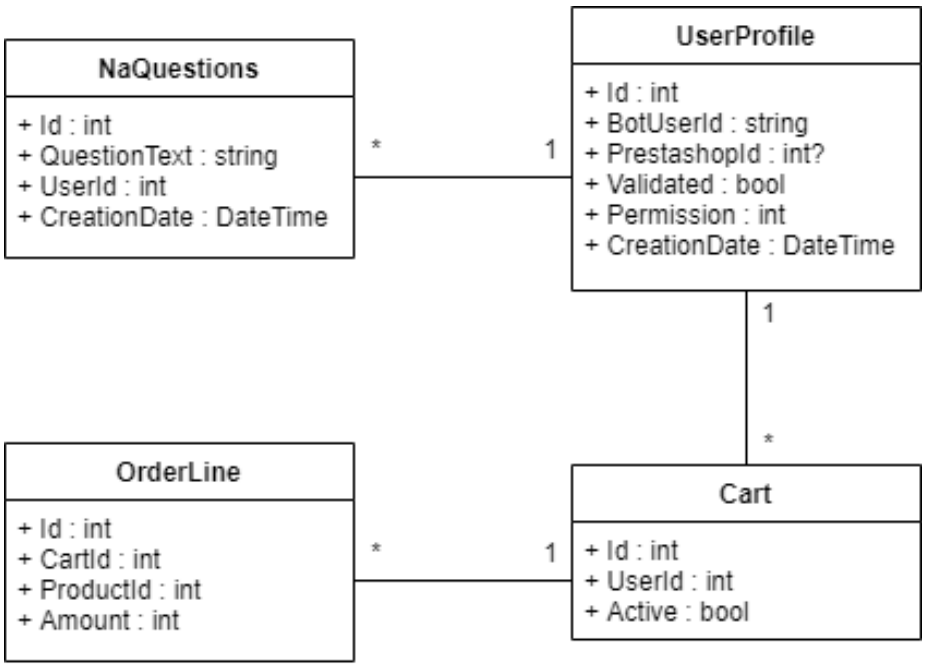


Figura 8.3: Diagrama de classes del bot

El diagrama de classes de l'aplicació és molt simple. La informació que necessita persistir dels usuaris és molt poca, la persistència de la informació relativa a les converses es fa directament a l'storage del bot.

Capítol 9

Implementació i proves

En aquest capítol explicaré les decisions principals d'implementació així com diversos problemes que han sortit durant el desenvolupament del projecte i les solucions que s'han proposat.

9.1 Implementació de l'aplicació

S'ha implementat una web app .NET core que té un total de 38 classes, i altres classes de dades (com els AdaptiveCards en JSON). Les classes estan organitzades en:

- El Bot
- Diàlegs
- Controllers dels models de la base de dades
- Classes estàtiques de mètodes d'extensions
- Models de la base de dades
- Entitats de l'API de Prestashop
- Interfícies genèriques de renderització d'AdaptiveCards
- L'API de Prestashop
- Classes per accedir als serveis cognitius
- Startup, per definir totes les classes pel Dependency Injection

9.1.1 Diàlegs

Els diàlegs són l'eina per donar funcionalitat al bot. Hi ha alguns diàlegs implementats que han quedat obsolets degut a canvis de requeriments durant el projecte. Aquests diàlegs són:

- CancelAndHelpDialog: aquest diàleg serveix com a superclasse d'altres diàlegs. Quan un diàleg extén aquesta classe, dona a l'usuari la possiblitat de cancelar (abortar completament un diàleg) i/o demanar ajuda al mig del flux de conversa.
- CardDialog: aquest diàleg serveix com a superclasse d'altres diàlegs. Quan un diàleg extén aquesta classe rep funcions i paràmetres per tractar els AdaptiveCards (inhabilitar-los).
- PermissionDialog: aquest diàleg serveix com a superclasse d'altres diàlegs. Quan un diàleg extén aquesta classe obté mètodes per comprovar si un usuari té permís per executar un diàleg.
- AddProductInfoDialog
- AskUserInfoDialog
- CartToOrderDialog
- ConfirmOrderDialog
- MainDialog
- OrderProductDialog
- TechnicalAssistanceDialog
- UserLoginDialog
- UserValidationDialog

9.1.2 Models de la base de dades

Els models són objectes que ens proporcionen la persistència de dades dels usuaris. Com he esmentat anteriorment, el diagrama de classes és molt simple. Els models són:

- UserProfile
- Cart
- OrderLine
- NaQuestions

9.1.3 Entitats de Prestashop

Els entities de Prestashop són les classes que utilitza el Prestashop per la persistència de les dades necessàries per realitzar les funcionalitats de botiga virtual. Aquest sistema de classes no l'he dissenyat jo, ja venia donat. He hagut de programar serialitzadors per poder extreure informació amb la API. Les entitats que he necessitat són:

- Product
- Customer
- Country
- Currency
- Cart
- Order
- Language

9.1.4 Controladors de model

Per interactuar amb la base de dades he dissenyat uns controladors de model. Com que les interaccions són poques, he triat no fer servir un disseny sencer MVC i he decidit fer controladors "tot en un". També he fet un controlador de notificacions que s'encarrega d'enviar les notificacions proactives del bot a un usuari.

- ProductController
- UserController
- PurchaseController
- NotifyController
- QuestionController

9.1.5 Interfícies de render d'AdaptiveCards

Per no haver de fer renderitzadors personalitzats per cada classe (models, entities) he dissenyat unes interfícies genèriques per renderitzar automàticament les classes que es mostran als usuaris en AdaptiveCards.

- IIdentifiable
- IAttachable
- IImageAttachable
- Carouselable
- ImageCarouselable

9.1.6 Mètodes d'extensions

Els mètodes d'extensions són eines molt útils per ajudar a desenvolupar un projecte. Un mètode d'extensió és un mètode estàtic que agrega una funcionalitat a un tipus. Per exemple, el C# implementa automàticament el List<int>, però jo podria dissenyar un mètode estàtic d'extensió per agregar un nou mètode a la classe List<int> i utilitzar-lo dins del meu codi. Les classes d'extensió dissenyades són les següents:

- ApiCallerExtensions
- DictionaryExtensions
- LuisResolutionExtensions
- PermissionExtensions
- StepContextExtensions

9.2 Sistema de permisos

Un dels objectius del projecte és establir un sistema de permisos. Un sistema de permisos és un sistema jeràrquic d'usuaris per limitar les funcionalitats que cada usuari té accés. S'han establert 6 nivells:

```
1  public enum PermissionLevels : int
2  {
3      [Description("Superuser")]
4      Superuser = 0,
5      [Description("Vitrosep")]
6      Vitrosep = 1,
7      [Description("Customer")]
8      Customer = 2,
9      [Description("Representative")]
10     Representative = 3,
11     [Description("Lead")]
12     Lead = 4,
13     [Description("Unregistered")]
14     Unregistered = 5
15 }
```

Listing 9.1: Nivells de permisos

També s'ha creat una classe **PermissionDialog** que extén `ComponentDialog` (la classe base dels diàlegs). Aquest té funcions per comprovar que l'usuari que faci servir el bot tingui permís per fer-lo servir.

```
1      protected async Task<DialogTurnResult>
           ↪ CheckPermissionStepAsync(WaterfallStepContext
           ↪ stepContext, CancellationToken cancellationToken)
```

```
2       {
3           var user = await UserController
4               .GetUserByBotIdAsync(
                    ↪ stepContext.Context.Activity.From.Id );
5           var userPermission = (int)PermissionLevel;
6           if (user != null && user.Permission <= userPermission)
7           {
8               return await stepContext.NextAsync(stepContext.Options,
                    ↪ cancellationToken);
9           }
10          else
11          {
12              await stepContext.Context
13                  .SendActivityAsync(MessageFactory.Text(noPermission),
                        ↪ cancellationToken);
14              return await stepContext.EndDialogAsync(null,
                    ↪ cancellationToken);
15          }
16      }
17
18      public PermissionLevels PermissionLevel { get; set; }
```

Listing 9.2: Nivells de permisos

Tots els diàlegs hereten aquesta propietat **PermissionLevels** la qual li hauran de donar valor, i la funció **CheckPermissionStepAsync** que serà el primer WaterfallStep de cada diàleg que extengui PermissionDialog. Per exemple, el constructor del diàleg OrderProductDialog tindria la següent informació:

```
1           AddDialog(new WaterfallDialog(nameof(WaterfallDialog), new
                ↪ WaterfallStep[]
2             {
3                 CheckPermissionStepAsync,
4                 LuisConversionStepAsync,
5                 ProductStepAsync,
6                 SelectionCardStepAsync,
7                 DisableCardStepAsync,
8                 AmountStepAsync,
9                 ConfirmLineStepAsync,
10                AddToCartStepAsync,
11            }));
12
13          InitialDialogId = nameof(WaterfallDialog);
14          PermissionLevel = PermissionLevels.Representative;
```

Listing 9.3: Ús de la restricció de permisos

Com veiem crida CheckPermissionStepAsync abans d'executar els altres WaterfallSteps i té el seu PermissionLevel definit.

De manera que si intentem utilitzar una funcionalitat per la qual no tenim permisos, veurem aquest missatge del bot:

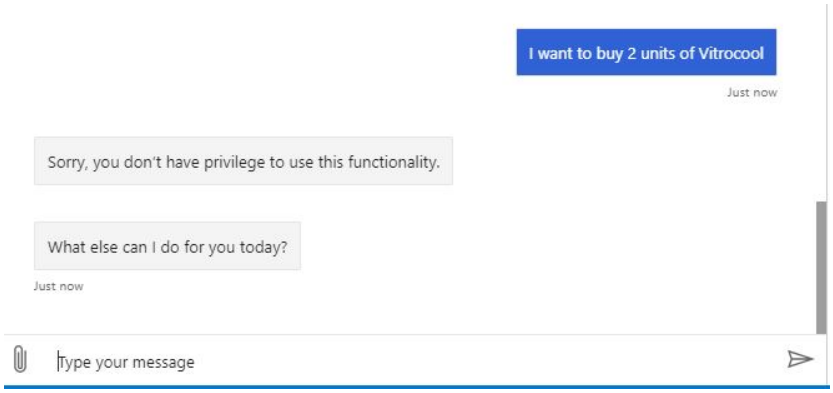


Figura 9.1: Missatge del bot a l'hora de comprovar permisos

9.3 Problema amb els botons d'Adaptive Cards

Anteriorment, hem explicat que un dels problemes principals dels botons d'AdaptiveCards és que l'usuari els pot utilitzar quan la conversa ja ha avançat des de que s'ha enviat l'AdaptiveCard. Això comporta un missatge no esperat en el flux de conversa i es llança una excepció d'error.

Aprofitant que es pot assignar un Id (string) als botons de les AdaptiveCards s'ha decidit establir un protocol.

- Tots els botons dels adaptiveCards hauran de tenir com a id "submitButton"+ un nombre. (submitButton1, submitButton2, submitButton3...)
- Per cada AdaptiveCard que contingui botons es generarà un GUID.

  Un GUID és un identificador únic universal, essencialment un nombre de 16 bytes del format: `3F2504E0-4F89-11D3-9A0C-0305E82C3301`.
- Quan es creï un AdaptiveCard (ja sigui amb JSON o dins del codi utilitzant l'SDK) s'haurà de transformar a json, i cada vegada que es trobi un "submitButton"es canviarà pel GUID generat. De manera que tots els botons de l'AdaptiveCard tindran com a Id el mateix GUID.
- Després de renderitzar un AdaptiveCard i enviar-lo a l'usuari i just abans de llegir la resposta de l'usuari s'invalidarà l'AdaptiveCard.

  Això es durà a terme amb un diccionari que porta el compte de tots els GUIDs utilitzats com a Id, cada vegada que s'envia un AdaptiveCard s'afegeix el GUID al diccionari.
- Utilitzant **Middleware** (recordem que és una acció que es realitza sempre al rebre un missatge i abans i/o després de gestionar-lo), quan arribi una resposta provinent d'un SubmitButton d'un AdaptiveCard es comprovarà si ja ha estat utilitzat o no. Al codi del bot, abans de cridar el MainDialog trobem aquestes línies.

```
1        else if (conversationData.DisabledCards.ContainsKey(
             ↪ (string)jobject["id"]) )
2        {
3            await turnContext.SendActivityAsync(
                 ↪ MessageFactory.Text(CardUtils.sameCardMsg),
                 ↪ cancellationToken );
4        }
```

Listing 9.4: Comprovació del GUID abans de gestionar el missatge

De manera que havent implementat això, si intentem utilitzar l'AdaptiveCard per entrar el password quan la conversa ja ha avançat, el bot ens tornarà el següent missatge i el missatge enviat per l'usuari (el submit) no es procesarà:

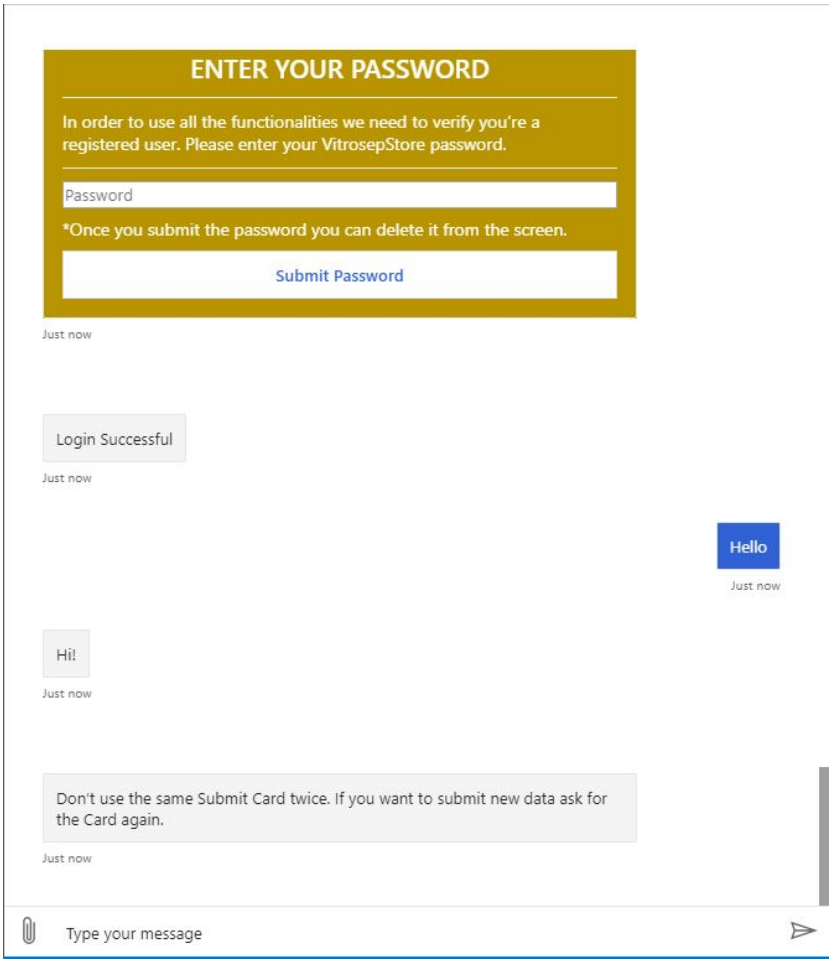


Figura 9.2: Missatge d'error a l'utilitzar un AdaptiveCard deshabilitat

Com podem veure he utilitzat l'AdaptiveCard per entrar el password, ha loguejat correctament. Seguidament li he dit Hello, el bot ha contestat i llavors he fet servir l'AdaptiveCard vell per enviar dades de nou.

L'implementació del sistema ha sigut correcte perquè no m'ho ha permès fer. M'ha enviat un missatge advertint-me i ha ignorat les dades que he enviat.

9.4 Reconeixement LUIS i WaterfallSteps

Al capítol 5 he parlat de com hi ha bots que obtenen les respostes step-by-step i hi ha bots que integren NLU, permetent obtenir informació dins d'una frase.

El que jo he fet és un híbrid, per augmentar la fiabilitat. He aplicat NLU per reconèixer l'intent de l'interacció amb l'usuari així com l'extracció dels entities que hi puguin haver a la frase. La idea és reconèixer el màxim d'informació en la petició inicial de l'usuari i anar demanant step-by-step la informació que falta.

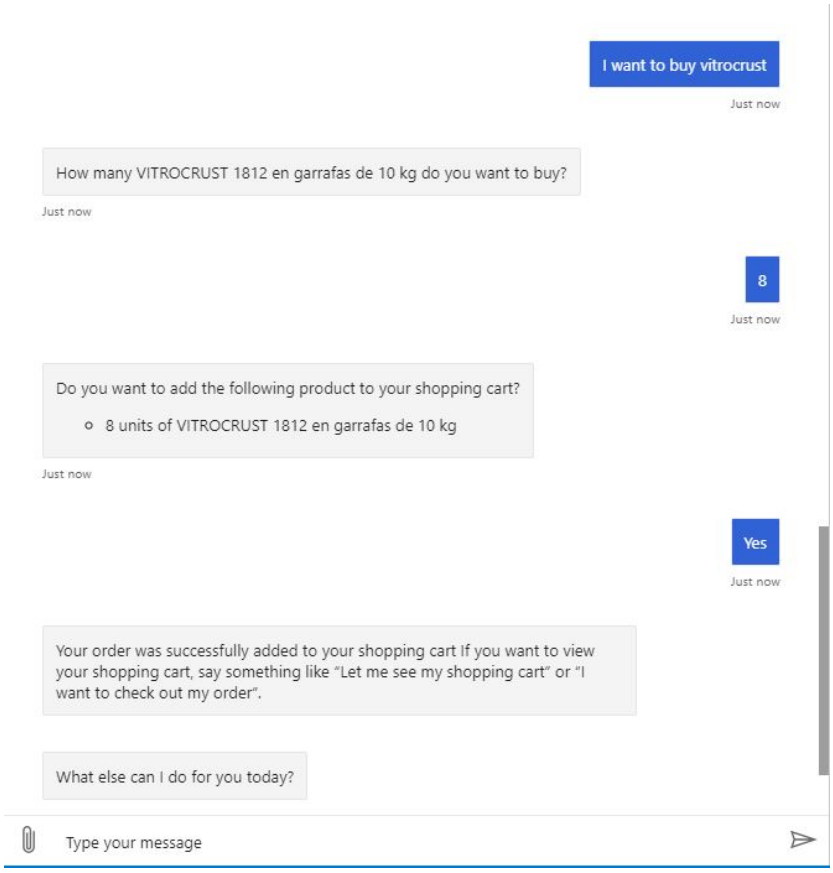


Figura 9.3: Diàleg de compra NLU + step by step

Com veiem al diàleg 9.3, l'usuari demana comprar un producte. El bot reconeix l'Intent de l'usuari i el producte que vol comprar, però veu que falta la quantitat a la frase. Així que, mitjançant una pregunta step-by-step demana la informació que falta fins que és capaç de construir una línia de comanda completa.

9.5 Missatges d'ajuda en tot moment

Una de les funcionalitats del bot és guiar a l'usuari en tot moment, oferint diàlegs d'ajuda si l'usuari així ho sol·licita.

Vaig dissenyar un sistema amb el qual l'usuari podria interrompre el flux de conversa en tot moment demanant *"help"*. El bot interrompria el flux de conversa, enviaria un missatge d'ajuda i reemprendria el flux de conversa.

Aquest era un missatge d'ajuda genèric, però per donar ajuda més eficaç vaig pensar en dissenyar missatges d'ajuda que es veiessin afectats pel context. És a dir, donar un missatge d'ajuda depenent de l'estat del flux de conversa.

Això va ser molt difícil d'implementar ja que vaig haver d'estudiar la gestió del flux de conversa i les variables que hi intervenen, navegant per totes les classes del repositori de l'SDK, sense cap guia.

Finalment vaig trobar una solució, tot i així només volia veure si era possible (s'ha d'escriure la base de coneixement dels diàlegs d'ajuda contextuals, però el procediment ja està fet).

Com veiem a la figura 9.4, el bot ha demanat a l'usuari que entri el correu. Quan l'usuari ha demnanat ajuda, en comptes de donar el missatge genèric d'ajuda li ha donat un missatge d'ajuda que depenia de l'estat del flux de conversa.

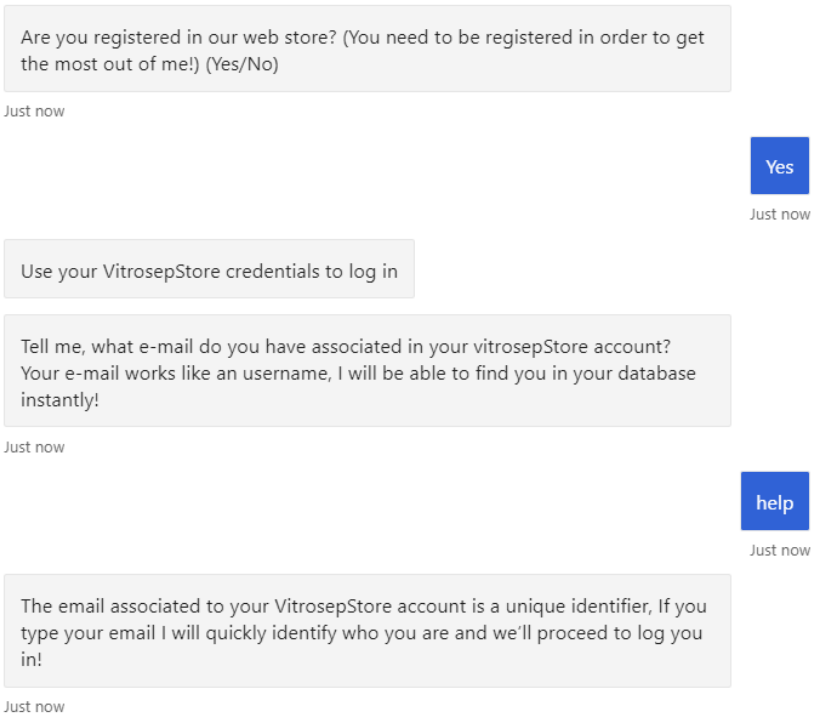


Figura 9.4: Exemple de missatge d'ajuda contextual

Capítol 10

Implantació i resultats

El sistema desenvolupat en aquest projecte està pendent de ser implantat. La data d'implantació a l'empresa és a **principis de Juliol del 2020**.

Abans ha de passar per una fase de proves, s'ampliarà la base de coneixement i es puliran alguns diàlegs.

Seguidament explicaré tot el necessari per a implantar el sistema. A més, posteriorment a la bibliografia, hi haurà un capítol anomenat Manuals, on explicaré com instal·lar els elements necessaris per fer funcionar el sistema.

10.1 Creació d'un bot a Azure

Primer hem de crear un bot a Azure. Per fer això necessitem un compte al portal d'Azure. https://azure.microsoft.com/es-es/features/azure-portal/

Un cop dins premem el botó *Crear un recurso* i sel·leccionem: IA Machine Learning -> Web App bot.

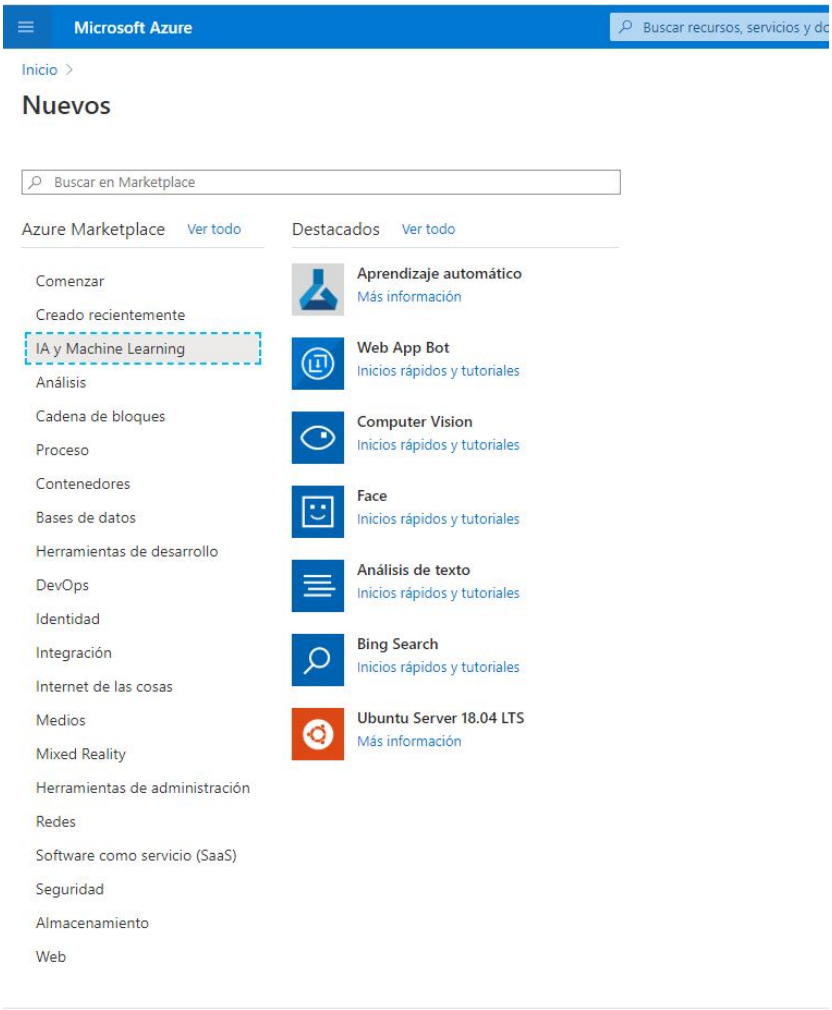


Figura 10.1: Menú per afegir el recurs Web App bot a Azure

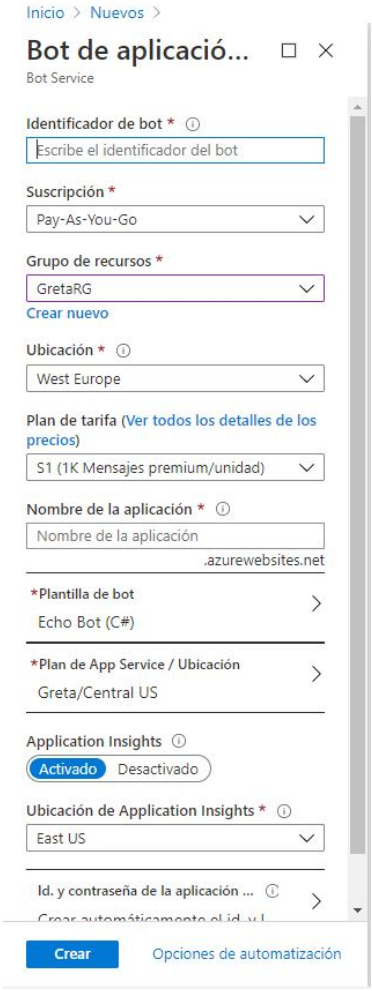


Figura 10.2: Emplenem la informació del bot

Seguidament, com veiem a la figura 10.2 emplenarem la informació del bot. Crearem un nou grup de recursos per afegir els serveis cognitius i altres recursos (base de dades, etc.) amb què s'hagi de comunicar i sel·leccionarem una subscripció.

Jo aquesta feina la vaig fer abans de programar el bot. El que això em va generar va ser un sample on poder començar a programar. El sample era un CoreBot és a dir, un bot que només rep missatges HTTP i contesta amb el mateix missatge que l'usuari ha enviat.

10.1.1 Vinculació amb els serveis LUIS i QnA

Els serveis LUIS i QnA tenen endpoint i claus privades. L'endpoint és una adreça on el bot pot consultar la base de coneixement via HTTP i les claus privades serveixen per realitzar l'autenticació.

Aquesta informació s'escriu dins l'arxiu de dades privat `appsettings.json` i després es pot recuperar mitjançan l'interfície `IConfiguration`. El següent codi extreu les dades de l'arxiu appsettings per carregar-les en les classes que farem servir per interactuar amb els serveis:

```
1  public class BotServices : IBotServices
2  {
3      public BotServices(IConfiguration configuration)
4      {
5          // Read the setting for cognitive services (LUIS) from the
               ↪ appsettings.json
6          Dispatch = new LuisRecognizer(new LuisApplication(
7              configuration["LuisAppId"],
8              configuration["LuisAPIKey"],
9              $"https://{configuration["LuisAPIHostName"]}
10 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣.api.cognitive.microsoft.com"),
11             new LuisPredictionOptions { IncludeAllIntents = true,
                   ↪ IncludeInstanceData = true },
12             true);
13
14         QnA = new QnAMaker(new QnAMakerEndpoint
15         {
16             KnowledgeBaseId = configuration["QnAKnowledgebaseId"],
17             EndpointKey = configuration["QnAEndpointKey"],
18             Host = configuration["QnAEndpointHostName"]
19         });
20     }
21
22     public LuisRecognizer Dispatch { get; private set; }
23
24     public QnAMaker QnA { get; private set; }
25 }
```

Listing 10.1: Ús de l'interfície IConfiguration per comunicar-se amb els serveis

I amb aquestes dues classes `LuisRecognizer` i `QnAMaker` (que utilitza la classe Bot), el bot és capaç de parlar amb els endpoints d'una manera programmer-friendly.

10.1.2 Publicació del bot

Com ja he dit, en el meu cas, primer vaig crear el bot a Azure, em vaig descarregar el sample i vaig començar a programar. D'aquesta manera s'han creat uns arxius dins del projecte que vinculen el projecte VisualStudio amb el bot d'Azure.

Si tenim aquesta configuració podem anar a Compilar -> Publicar Bot.

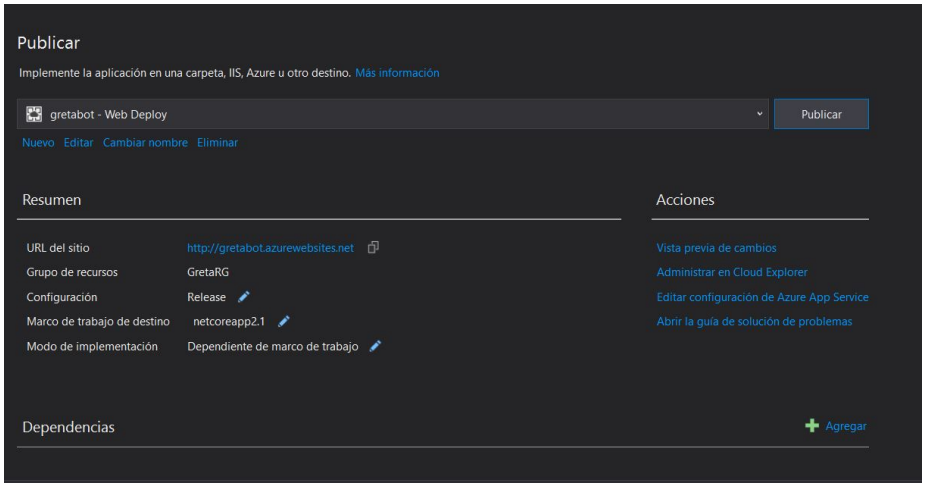


Figura 10.3: Menú de compilació per publicar el bot a Azure

10.1.3 Vincular el bot d'Azure amb un canal

Per fer l'exemple faré servir el Telegram. El Telegram té una gestió de bots peculiar, ja que es fa a través d'un bot: **El BotFather**.

Per crear un bot a Telegram (recordem que aquest només serà la interfície usuari que parlarà amb el nostre ChatBot que hem hostejat a Azure) hem d'iniciar una conversa amb el BotFather i seguir els següents passos:

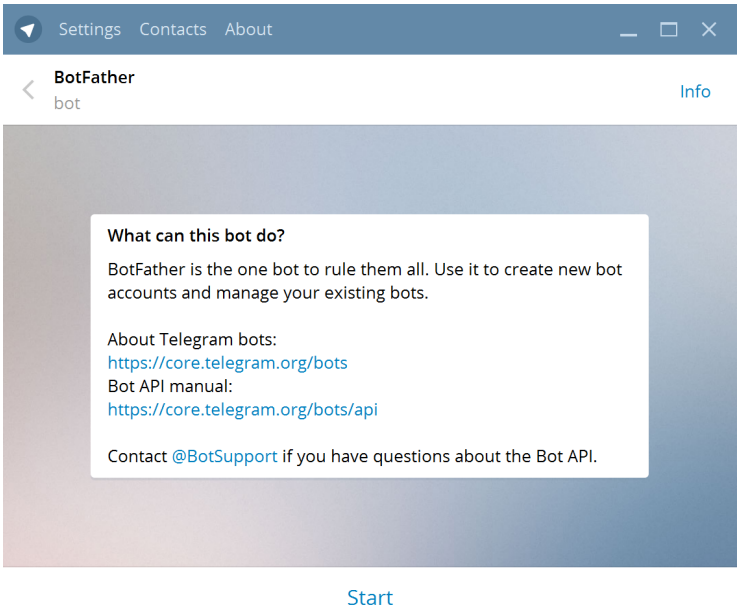


Figura 10.4: Primera interacció amb el BotFather

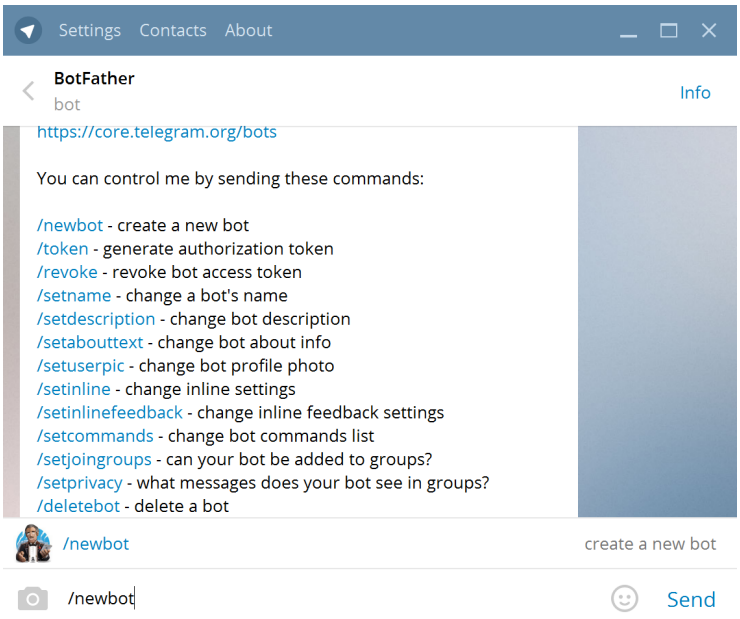


Figura 10.5: Creem el bot mitjançant la comanda newbot

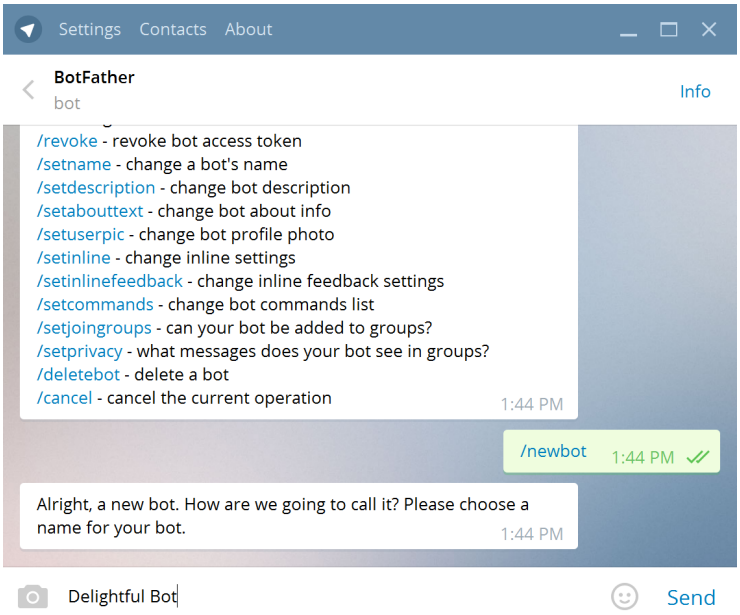


Figura 10.6: El BotFather ens demana que triem el nom pel qual ens coneixeran els usuaris

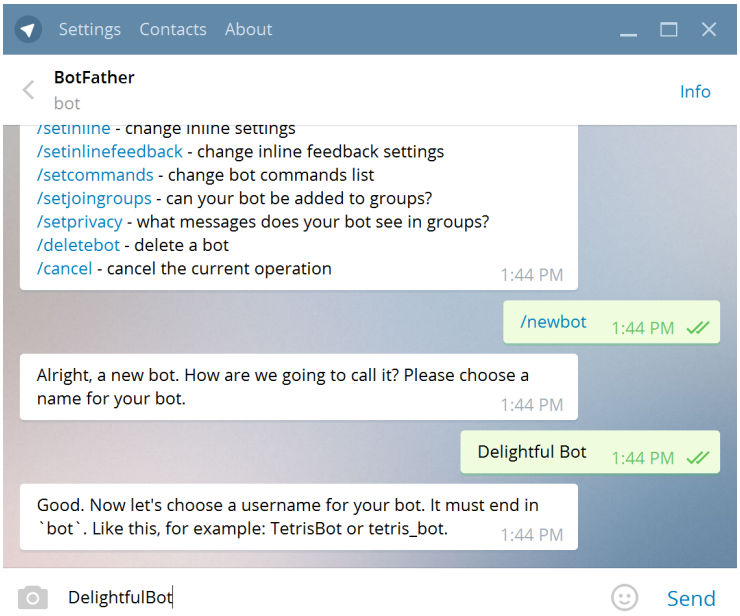


Figura 10.7: Triem el nom del nostre bot

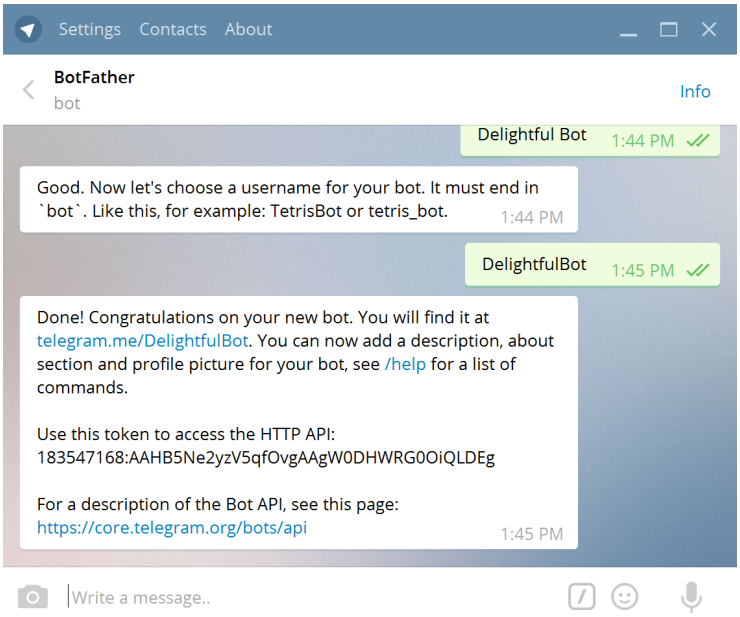


Figura 10.8: El BotFather ens dona el token del bot

Un cop hem obtingut el token del bot, ja podrem vincular-lo amb el ChatBot que hem desenvolupat i hostejat a Azure. Aquest token és l'identificador del bot a telegram.

Ens hem de dirigir al nostre ChatBot al portal d'Azure, seleccionar la vinculació a canals, triar telegram i entrar el token que ens ha donat el BotFather.

A partir d'aquí quan algú enviï un missatge al bot de Telegram, el ChatBot que hem hostejat a Azure el rebrà i contestarà igual com si féssim servir l'emulador.

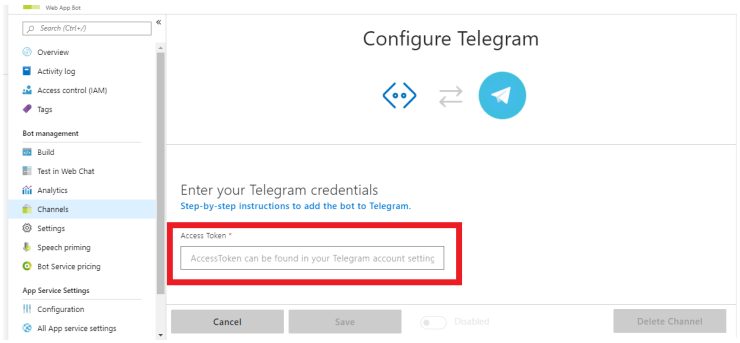


Figura 10.9: Introducció del token de Telegram a Azure

Capítol 11

Conclusions

Aquest projecte ha sigut un gran repte i una gran oportunitat que m'ha donat l'empresa per dissenyar una aplicació d'aquestes dimensions i tenir el privilegi de veure-la implantada el mes que ve. Sumant a tot això el poder participar en una de les tecnologies de l'avantguarda del màrqueting i l'activitat empresarial, com és el món dels ChatBots.

Ha sigut una immersió en un llenguatge, tecnologia i framework nous que no havia vist durant la carrera, però alhora he pogut aplicar molts conceptes i metodologies apreses.

La feina de quasi bé un any que es veu reflectida en més de 15mil línies de codi i una base de coneixement per emplenar sense límits, que ha comportat moltíssima recerca amb un abast d'informació menys que escàs a internet. Per sort un equip de professionals i developers del Microsoft Bot Framework repassava l'StackOverflow tres cops per setmana i podia aclarir dubtes allà en cas d'emergència.

Un dels grans objectius personals que tenia en aquest projecte era testejar els límits del framework, veure què podia fer i què no i com dissenyar-ho. Mantenir un nucli de l'aplicació robust però permetent que altres treballadors de l'empresa puguin col·laborar al projecte sense tenir cap experiència en la informàtica, poguent gestionar el contingut des del mateix software de gestió de l'empresa (SAP), com des de panells al navegador, ampliant la base de coneixement.

La planificació inicial del projecte ha sigut una aposta a cegues, ja que desconeixent totalment les eines que tenia no podia planificar de manera clara la feina que faria en el futur. Tot i així, gràcies a la metodologia flexible escollida, m'ha permès planejar el treball a curt termini i continuar avançant de forma constant en el projecte.

En definitiva, aquest projecte ha sigut un gran començament en la meva carrera laboral que m'ha donat experiència en una tecnologia emergent, que sempre és un gran punt positiu. A més a més, tinc el privilegi de continuar treballant en aquest projecte un cop estigui implantat, ampliant la base de coneixement i les funcionalitats, adaptant-ho a les noves necessitats de l'empresa.

Capítol 12

Treball Futur

El projecte tal i com està ara és una versió quasi completa del producte final. D'aquí fins el dia de la implantació a **principis de Juliol de 2020** estaré treballant per acabar els últims aspectes inacabats i animar a altres col·laboradors a ampliar la base de coneixement.

Els principals objectius del futur són:

- Completar la funcionalitat permetent que les comandes realitzades arribin a Prestashop.
- Ampliar la base de coneixement.
- Millora dels missatges enviats pel bot (petites modificacions per reduir text i millorar comprensió)
- Afegir funcionalitats com:
  - Sistema de notificacions per ofertar productes als clients
  - Sistema CRM pels treballadors de l'empresa del departament de màrqueting
- Estudi de la manera que tenen els clients d'interactuar amb el bot per millorar la base NLU.
- Sistema de comprovació de les comandes, per revisar que els recanvis que comprin els clients siguin per les seves màquines (que no s'equivoquin comprant productes amb una foto semblant que no els hi serviran, ha passat anteriorment a l'empresa)
- Sistema de gestió de magatzem pels treballadors del taller

Bibliografia

[1] BrightDevelopers. Bot Framework v4 Dialog Prompt and Waterfall. https://www.brightdevelopers.com/bot-framework-v4-dialog-prompt-and-waterfall/, 2018. Accedit: 08/08/2019.

[2] Entity Framework Tutorial. Entity Framework Core. https://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx, 2017. Accedit: 10/11/2019.

[3] inConcert. Qué es NLU y cuál es relación con los ChatBots. https://blog.inconcertcc.com/qu-en-nlu-y-cul-es-su-relacin-con-los-chatbots/, 2017. Accedit: 23/09/2019.

[4] Microsoft. AdaptiveCards: Schema Explorer. https://adaptivecards.io/explorer/, 2019. Accedit: 16/10/2019.

[5] Microsoft Docs. Connect a bot to Telegram. https://docs.microsoft.com/en-us/azure/bot-service/bot-service-channel-connect-telegram?view=azure-bot-service-4.0, 2019. Accedit: 01/02/2020.

[6] Microsoft Docs. Use multiple LUIS and QnA models. https://docs.microsoft.com/en-us/azure/bot-service/bot-builder-tutorial-dispatch?view=azure-bot-service-4.0&tabs=cs, 2019. Accedit: 9/10/2019.

[7] Microsoft Docs. Tutorial: Create and deploy a basic bot. https://docs.microsoft.com/en-us/azure/bot-service/bot-builder-tutorial-basic-deploy?view=azure-bot-service-4.0&tabs=csharp, 2019. Accedit: 5/8/2019.

[8] Prestashop Forum. Prestashop: What is secure key? How to import secure key to DB. https://www.prestashop.com/forums/topic/43275-solved-what-is-secure_key-how-to-import-current-sql-db-into-ps/, 2010. Accedit: 9/01/2020.

[9] SciSharp. BotSharp: The Open Source AI Chatbot Platform Builder. https://github.com/SciSharp/BotSharp, 2019. Accedit: 4/08/2019.

[10] StackOverflow. How to work properly with advanced Adaptive Cards? https://stackoverflow.com/questions/58389870/how-to-work-properly-with-advanced-adaptive-cards, 2019. Accedit: 15/10/2019.

[11] StackOverflow. Defining an API caller class with ASP.NET Core. https://stackoverflow.com/questions/58710831/defining-an-api-caller-class-with-asp-net-core, 2019. Accedit: 5/11/2019.

[12] StackOverflow. How to authenticate Prestashop API REST Client. https://stackoverflow.com/questions/58952427/how-to-authenticate-prestashop-api-rest-client, 2019. Accedit: 21/11/2019.

[13] StackOverflow. Working around the lack of support for Adaptive Cards 1.2. https://stackoverflow.com/questions/58487356/working-around-the-lack-of-support-for-adaptive-cards-1-2, 2019. Accedit: 21/10/2019.

[14] StackOverflow. How to use State Accessors to get properties in Bot Framework. https://stackoverflow.com/questions/58338918/how-to-use-state-accessors-to-get-properties-in-bot-framework, 2019. Accedit: 11/10/2019.

[15] StackOverflow. Bot Framework Emulator: Unknown Host. https://stackoverflow.com/questions/57669354/bot-framework-emulator-unknown-host, 2019. Accedit: 27/08/2019.

[16] StackOverflow. Best key to store UserState in external DB. https://stackoverflow.com/questions/59894570/best-key-to-store-userstate-in-external-db, 2020. Accedit: 24/01/2020.

[17] StackOverflow. Custom help messages for every WaterfallStep. https://stackoverflow.com/questions/59767240/custom-help-messages-for-every-waterfallstep, 2020. Accedit: 16/01/2020.

[18] StackOverflow. EF Core table first not saving entities in the database. https://stackoverflow.com/questions/61770329/ef-core-table-first-not-saving-entities-in-the-database, 2020. Accedit: 13/05/2020.

Capítol 13

Manuals

13.1 Manuals d'instal·lació

Els manuals d'instal·lació mostren aquells passos necessaris per a implantar el sistema.

13.1.1 Instal·lació de l'entorn de treball

Per executar el projecte necessitarem tres elements bàsics:

- Net Core 2.1
- Visual Studio 2019
- El Bot Framework Emulator

Visual Studio 2019

El Visual Studio 2019 ens descarregarem la versió Communty del següent link:

- https://visualstudio.microsoft.com/es/downloads/

Dotnet CLI

Alternativament també ens podem instal·lar el CLI de .NET per executar el projecte, des d'aquest link:

- https://docs.microsoft.com/en-us/dotnet/core/tools/?tabs=netcore2x

Utilitzant el CLI podem navegar fins on haguem descarregat el projecte i executar `dotnet run`.

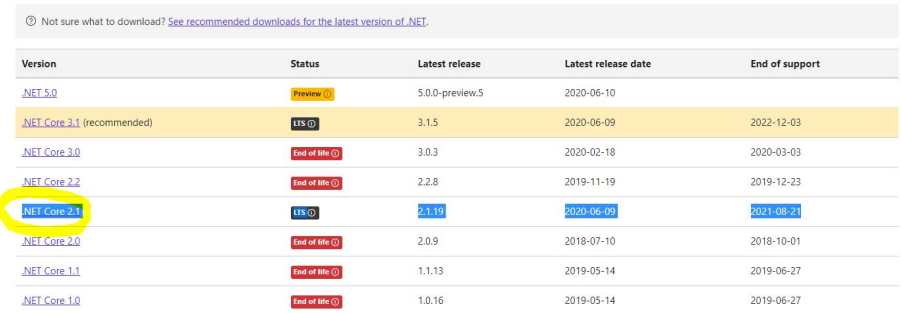
Net Core 2.1

El .NET core 2.1 ens el descarregarem seguint els següents passos:

1. Accedim al següent link:
   - https://dotnet.microsoft.com/download
2. Anem a .NET core downloads -> All .NET core downloads

Figura 13.1: Accedim a All .NET core downloads

3. Sel·leccionem el .NET core 2.1 (El Bot Framework no suporta el 3.1)

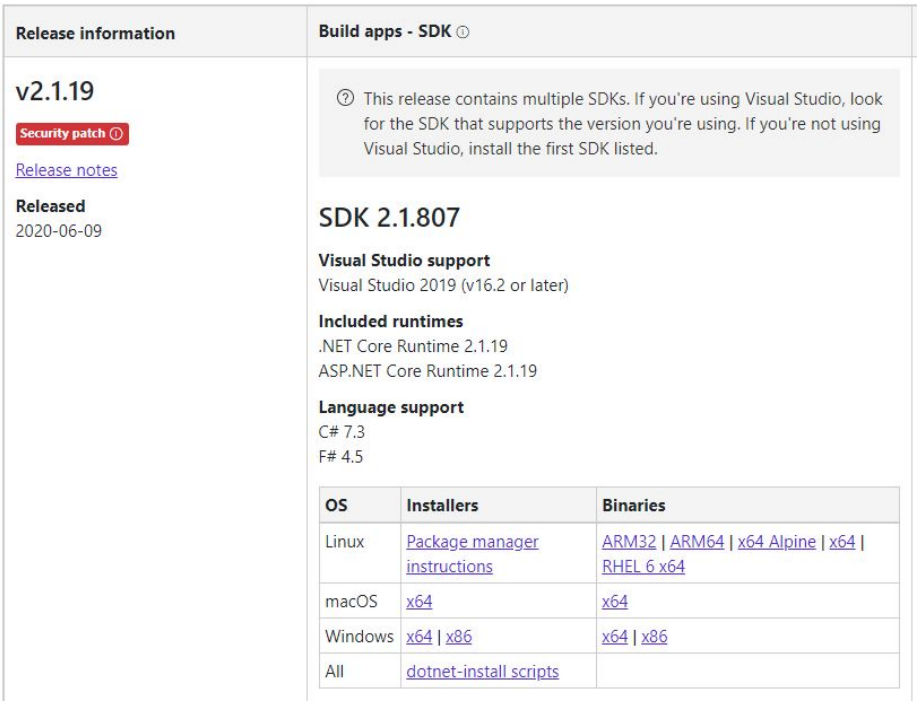


Not sure what to download? See recommended downloads for the latest version of .NET.

Version	Status	Latest release	Latest release date	End of support
.NET 5.0	Preview	5.0.0-preview.5	2020-06-10	
.NET Core 3.1 (recommended)	LTS	3.1.5	2020-06-09	2022-12-03
.NET Core 3.0	End of life	3.0.3	2020-02-18	2020-03-03
.NET Core 2.2	End of life	2.2.8	2019-11-19	2019-12-23
.NET Core 2.1	LTS	2.1.19	2020-06-09	2021-08-21
.NET Core 2.0	End of life	2.0.9	2018-07-10	2018-10-01
.NET Core 1.1	End of life	1.1.13	2019-05-14	2019-06-27
.NET Core 1.0	End of life	1.0.16	2019-05-14	2019-06-27

Figura 13.2: Ens descarreguem el .NET core 2.1

4. Seleccionem l'última versió de l'SKD i l'instal·lem pel nostre sistema operatiu.



Release information

v2.1.19

Security patch

Release notes

Released
2020-06-09

Build apps - SDK

This release contains multiple SDKs. If you're using Visual Studio, look for the SDK that supports the version you're using. If you're not using Visual Studio, install the first SDK listed.

SDK 2.1.807

Visual Studio support
Visual Studio 2019 (v16.2 or later)

Included runtimes
.NET Core Runtime 2.1.19
ASP.NET Core Runtime 2.1.19

Language support
C# 7.3
F# 4.5

OS	Installers	Binaries
Linux	Package manager instructions	ARM32 | ARM64 | x64 Alpine | x64 | RHEL 6 x64
macOS	x64	x64
Windows	x64 | x86	x64 | x86
All	dotnet-install scripts	

Figura 13.3: Sel·leccionem l'última versió de l'SDK

Bot Framework Emulator

Ens descarreguem l'última release del Bot Framework Emulator del següent link:

- https://github.com/microsoft/BotFramework-Emulator/releases

Instal·lem la versió pel nostre sistema operatiu.

13.2 Instal·lació del codi del projecte

El codi del projecte està penjat a GitHub, on també podreu fer un seguiment del meu treball i veure la nova feina que s'ha anat incorporant als Pull Requests. El projecte el podreu trobar al següent link:

- https://github.com/fcapallera/Greta

MOLT IMPORTANT

El projecte no està complert ja que hi ha un arxiu amb dades sensibles que no es pot penjar a internet. Si voleu executar el projecte m'haureu de demanar l'arxiu `appsettings.json` per correu (fcapallera@gmail.com) i enganxar-lo a la carpeta arrel del projecte.

13.3 Execució del programa

Quan s'hagi obert el projecte amb Visual Studio, començarà a descarregar totes les llibreries utilitzades pel projecte.

Un cop estigui tot llest podrem provar l'aplicació:

1. Obrim el Visual Studio i executem l'aplicació, això ens obrirà una pàgina del navegador, la podem ignorar.
2. Obrim el Bot Framework Emulator
3. Apretem **Open Bot** a la pestanya **Welcome** i introduïm la següent configuració:

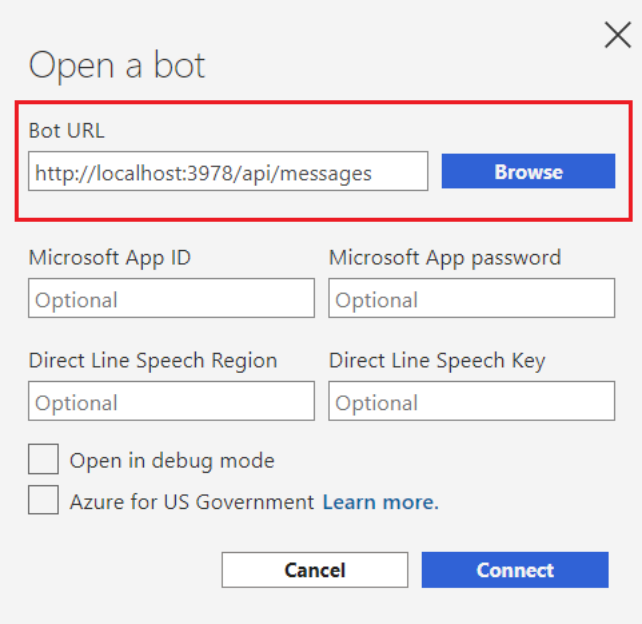


Figura 13.4: Configuració del bot

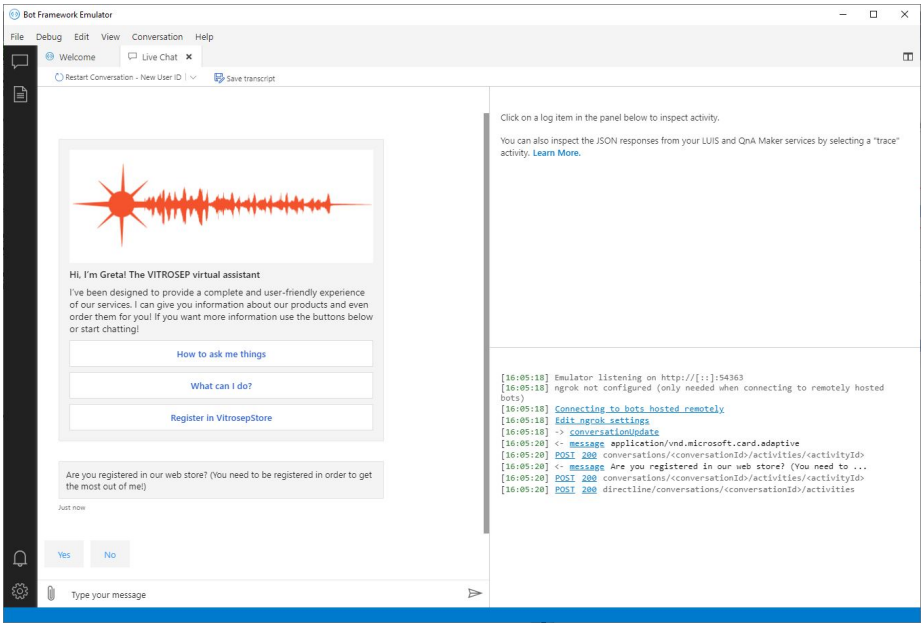


Figura 13.5: Vista de l'emulador quan obrim el bot

4. Seguirem el flux de diàleg fins que demani les credencials. Per autenticar-nos farem servir:

   - **Correu:** fcapallera@gmail.com
   - **Password:** ProjectGretaTFG

     Aquest perfil seguirà funcionant fins el dia de la defensa del tfg i després s'invalidarà.

5. A partir d'aquí podem interactuar amb la limitada base de coneixement, o utilitzar alguna de les funcionalitats.

   - I want to buy [nombre] units of [nom del producte] (o altres maneres que se us acudeixin de demanar un producte).
   - I want to confirm my order / I want to finish shopping
   - I want to validate [nom d'un usuari] / I want to validate a customer
   - I need help with my machine / Something is wrong with my machine