



Διπλωματική Εργασία

**Αλληλεπίδραση με
conversational interface και
ανάπτυξη ενός bot για σύμβουλο
δραστηριοτήτων**

Μπάκας Δημήτριος
Α.Μ. : 235704

*Η παρούσα εργασία εκπονήθηκε σε συνεργασία με το Εργαστήριο
Γραφικών πολυμέσων και Γεωγραφικών Συστημάτων του Τμήματος
Μηχανικών Η/Υ & Πληροφορικής.*



Περίληψη

Αυτή η διπλωματική εργασία έχει ως στόχο να προσφέρει μια πλατφόρμα επικοινωνίας μεταξύ χρήστη και υπολογιστή. Η χρήση αυτής της πλατφόρμας είναι ως ένας σύμβουλος δραστηριοτήτων ο οποίος συνομιλεί με το χρήστη και αφότου έχει συλλέξει αρκετά στοιχεία για να μπορούν να παρθούν βάσιμα συμπεράσματα, αρχίζει να του παρουσιάζει συμβουλές με τις οποίες ο χρήστης μπορεί να βελτιώσει τη ποιότητα της ζωής του. Αυτό επιτυγχάνεται με τη δημιουργία ενός chatbot το οποίο καταφέρνει να επικοινωνήσει με το χρήστη σε φυσική γλώσσα, δίνοντας στο χρήστη την αίσθηση ότι συνομιλεί με κάποιο άνθρωπο και όχι με ένα υπολογιστή. Φυσικά το chatbot ανακοινώνει εξ αρχής στο χρήστη ότι δεν είναι πραγματικός άνθρωπος, ούτως ώστε ο χρήστης να μπορεί να ανταποκριθεί αναλόγως. Πέρα από τη φυσική ομιλία και τη προσφορά συμβουλών, στόχος της διπλωματικής εργασίας είναι και να υπάρχει εύκολα η δυνατότητα επέκτασης τόσο των ερωτήσεων που απαντά ο χρήστης, όσο και των συμβουλών που προσφέρονται. Πράγμα το οποίο μπορεί να πραγματοποιηθεί κατευθείαν μέσα από τη πλατφόρμα από άτομα τα οποία είναι διαχειριστές.

Abstract

This dissertation aims to provide a communication platform between user and computer. The use of this platform is as an activity coach who talks to the user and after it has gathered enough information to be able to draw sound conclusions, it begins to present tips, with which the user can improve his quality of life. This is achieved by creating a chatbot that manages to communicate with the user in natural language, giving the user the feeling that he is talking to a person and not to a computer. Of course, the chatbot informs the user from the start of the conversation that it is not a real person, so that the user can respond accordingly. In addition to the use of natural language and the offer of advice, another aim of the dissertation is to have the possibility of easily extending, both the questions answered by the user and the tips that are offered. Which can be done directly through the platform by users who are administrators.

Πίνακας Περιεχομένων

Περίληψη	3
Abstract	4
Πίνακας Περιεχομένων	5
Πίνακας Σχημάτων	7
1. Εισαγωγή	9
1.1. Bots	9
1.1.1. CHATBOTS	9
1.1.2. VIRTUAL ASSISTANTS	9
1.2. Turing Test	10
1.3. Artificial Intelligence	10
1.3.1. MACHINE LEARNING	10
1.4. Frameworks	10
1.4.1. AZURE BOT SERVICE	11
1.4.2. CLUSTERING	11
1.4.3. RULES ENGINE	12
2. Ανάπτυξη conversational bot χρησιμοποιώντας Azure Bot Service ...	14
2.1. LUIS	14
2.2. Σύνδεση QnA Maker	16
2.3. Dialog	16
2.4. Σύνδεση bot με βάση δεδομένων (Azure Cosmos DB)	17
2.5. Clustering με ML.NET	18
2.6. Εφαρμογή Rules Engine με NRules	19
2.7. Ενεργοποίηση proactive μηνμάτων	19
2.8. Publication	20

2.9.Επιλογή Καναλιών.....	21
3. Εγχειρίδιο Χρήσης / Εγκατάστασης.....	23
4. Αρχιτεκτονική / Υλοποίηση	33
4.1. Αρχιτεκτονική	33
4.1.1. LUIS	34
4.1.2. QNA MAKER.....	35
4.1.3. AZURE COSMOS DB.....	36
4.1.4. ML.NET.....	37
✓ <i>K-means</i> 38	
4.1.5. NRULES.....	38
✓ <i>Rete</i> 39	
4.1.6. AZURE FUNCTIONS.....	40
4.1.7. PROACTIVE MESSAGES	40
4.1.8. ADAPTIVE CARDS	40
4.2.Εργαλεία / Λογισμικό	41
4.3.Υλοποίηση.....	41
4.3.1. MAINDIALOG.CS	43
4.3.2. PERSONALDETAILSDIALOG.CS.....	46
4.3.3. QUESTIONNAIRECHOICEDIALOG.CS	48
4.3.4. TOPFIVEDIALOG.CS.....	48
4.3.5. REENTERDETAILSDIALOG.CS.....	50
4.3.6. AUTHENTICATIONDIALOG.CS.....	50
4.3.7. UPLOADTIPSORQUESTIONNAIRESDIALOG.CS	51
4.3.8. NUMBEROFTIPSDIALOG.CS	51
4.3.9. UPLOADTIPSDIALOG.CS	51
4.3.10. NAMEOFQUESTIONNAIREDIALOG.CS	52
4.3.11. UPLOADQUESTIONNAIRESDIALOG.CS.....	53
4.3.12. CANCELANDHELPDIALOG.CS.....	54
4.3.13. PERSONALDETAILS.CS.....	55
4.3.14. RESPONSETEXT.CS	56
4.3.15. QUESTIONTOPFIVE.CS	57
4.3.16. TIP.CS.....	58
4.3.17. TIPCHOOSERULE.CS.....	58
4.3.18. NOTIFYCONTROLLER.CS.....	59
4.3.19. DIALOGBOT.CS	60
4.3.20. ADAPTIVECARDPROMPT.CS.....	61
4.3.21. PREPARECARD.CS.....	61
4.3.22. LUISMODEL.CS.....	61
4.3.23. STARTUP.CS.....	62
4.3.24. CONNECTIONRECOGNIZER.CS.....	62
4.3.25. NUGET PACKAGES.....	63
4.3.26. AZURE FUNCTION	63
4.3.27. DATABASE – AZURE COSMOS DB.....	64
4.3.28. LUIS (LANGUAGE UNDERSTANDING)	64
4.3.29. QNA MAKER.....	65
5. Βιβλιογραφία	66

Πίνακας Σχημάτων

Σχήμα 1: Δημιουργία Intent	15
Σχήμα 2: Καταχώρηση Intent.....	15
Σχήμα 3: Δημιουργία Entity	15
Σχήμα 4: Review Endpoint Utterances.....	15
Σχήμα 5: Train, Test και Publish to LUIS μοντέλο.....	16
Σχήμα 6: Δημιουργία Chit-chat	16
Σχήμα 7: Βάση Δεδομένων μέσω του Data Explorer	18
Σχήμα 8: Σύνδεση Deployment Center με GitHub	20
Σχήμα 9: Επιλογή Καινούριου Καναλιού.....	21
Σχήμα 10 : Κανάλια που Χρησιμοποιεί το bot.....	22
Σχήμα 11: Καλωσοριστικό Μήνυμα.....	23
Σχήμα 12: Ερώτηση Ηλικίας.....	24
Σχήμα 13: Ερώτηση Φύλου.....	24
Σχήμα 14: Ερώτηση Καπνιστή.....	24
Σχήμα 15: Ερώτηση Κατανάλωσης Νερού και Ύπνου	24
Σχήμα 16: Ερώτηση Φυσικής Άσκησης	25
Σχήμα 17: Επικύρωση Στοιχείων	25
Σχήμα 18: Επιλογή Ερωτηματολογίου	25
Σχήμα 19: Παρουσίαση Ερωτήσεων	26
Σχήμα 20: Έξοδος Διαλόγου	26
Σχήμα 21: Απάντηση Ερωτηματολογίων	27
Σχήμα 22: Αποτελέσματα Personality Traits	27
Σχήμα 23: Ολοκληρωμένο Ερωτηματολόγιο.....	28
Σχήμα 24: Tip.....	28
Σχήμα 25: Αλλαγή Προσωπικών Δεδομένων	28
Σχήμα 26: Εισαγωγή Κωδικού Διαχειριστή.....	29
Σχήμα 27: Αριθμός Tips προς Αποστολή στη Βάση Δεδομένων	29
Σχήμα 28: Κάρτα Εισαγωγής Tip	30
Σχήμα 29: Αριθμός Ερωτήσεων στο Ερωτηματολόγιο	30
Σχήμα 30: Κάρτα Εισαγωγής Ερώτησης	31
Σχήμα 31: Ερώτηση Ρουτίνας (Chit-chat)	32
Σχήμα 32: Μήνυμα Βοήθειας	32
Σχήμα 33: Σχεδιάγραμμα Λειτουργίας Τεχνολογιών.....	34
Σχήμα 34: LUIS Response τύπου JSON	35
Σχήμα 35: bot-storage Container	37
Σχήμα 36: tips Container	37
Σχήμα 37: questionnaires Container.....	37
Σχήμα 38: Αλγόριθμος Rete	39
Σχήμα 39: Flowchart Λειτουργίας Bot	42
Σχήμα 40: LUIS Intent Switch	44
Σχήμα 41: Αποστολή Προσωπικών Δεδομένων στη Βάση Δεδομένων.....	45
Σχήμα 42: Συλλογή Personality Traits για Clustering από τη Βάση Δεδομένων.....	45
Σχήμα 43: Υλοποίηση Clustering	46
Σχήμα 44: Text Prompt Validator.....	47
Σχήμα 45: Επιλογή Ερωτηματολογίου	48
Σχήμα 46: Επαν-έναρξη του διαλόγου TopFiveDialog	48

Σχήμα 47: Υπολογισμός Score Personality Traits.....	49
Σχήμα 48: Έλεγχος αν το Ερωτηματολόγιο έχει ήδη Ολοκληρωθεί	49
Σχήμα 49: Κάρτα Εισαγωγής Κωδικού Διαχειριστή στον Emulator.....	50
Σχήμα 50: Μετατροπή Κωδικού σε MD5 Hash.....	51
Σχήμα 51: Μηχανισμός Loop για Εμφάνιση των Καρτών Εισαγωγής Tips	51
Σχήμα 52: Κάρτα Εισαγωγής Tip στον Emulator.....	52
Σχήμα 53: Μηχανισμός Loop για Εμφάνιση των Καρτών Εισαγωγής Ερωτηματολογίων	53
Σχήμα 54: Κάρτα Εισαγωγής Ερώτησης στον Emulator	54
Σχήμα 55: Απόδοση των Default Απαντήσεων σε Ερώτηση.....	54
Σχήμα 56: Λειτουργία Βοήθειας και Εξόδου από Διάλογο.....	55
Σχήμα 57: Έλεγχος για NULL Κείμενο Εισόδου	55
Σχήμα 58: Λίστα Μηνυμάτων Χαιρετισμού.....	56
Σχήμα 59: Δημιουργία Μηνύματος Χαιρετισμού	56
Σχήμα 60: Λειτουργία Rules Engine.....	57
Σχήμα 61: Κανόνες για το Rules Engine.....	59
Σχήμα 62: Αποστολή Μηνύματος (Tip) σε κάθε Conversation Reference.....	59
Σχήμα 63: Επιλογή Μηνύματος (Tip) και Απόδοσή του στο Χρήστη	60
Σχήμα 64: Συλλογή Δεδομένων Χρήστη από το Κανάλι και Εκκίνηση Tasks.....	60
Σχήμα 65: Καταχώριση του Conversation Reference του Χρήστη.....	61
Σχήμα 66: Διάβασμα JSON από Adaptive Card.....	61
Σχήμα 67: Διαθέσιμα Intents	62
Σχήμα 68: Built-in Entities	62
Σχήμα 69: Σύνδεση QnA Maker	62
Σχήμα 70: Azure Function Flowchart.....	63
Σχήμα 71: Get Request στο Endpoint από το Azure Function	64

1. Εισαγωγή

1.1. Bots

Η αυτοματοποίηση έχει πια κατακτήσει ένα μεγάλο μέρος της ζωής μας. Έχουμε συνηθίσει να κατεβάζουμε και να παραμετροποιούμε προγράμματα για να αναλαμβάνουν συνηθισμένες εργασίες της καθημερινότητάς μας. Τι θα γινόταν αν αναθέταμε αυτή την υπευθυνότητα σε κάποιον άλλο. Αυτό το πρόβλημα έρχονται να μας λύσουν τα chatbots, με στόχο τους, να εκτελούν τις επιθυμίες του χρήστη, χωρίς να καταστρέφουν την ψευδαίσθηση επικοινωνίας του χρήστη με έναν άνθρωπο, αντί με άλλο ένα πρόγραμμα.

1.1.1. Chatbots

Το IM bot είναι ένα πρόγραμμα chatterbot (δηλαδή chatbot) που χρησιμοποιεί άμεσα μηνύματα (Instant Messages, IM) ως διεπαφή εφαρμογής. Οι χρήστες της συνομιλίας μπορούν να προσθέσουν το όνομα του IM bot στη λίστα φίλων τους με τον ίδιο τρόπο που προσθέτουν συναδέλφους, οικογένεια και φίλους. Επειδή το bot χρησιμοποιεί τεχνητή νοημοσύνη (AI), ο τελικός χρήστης έχει την αίσθηση ότι μιλάει με ένα πραγματικό άτομο και είναι ικανός να ξεχάσει ότι πραγματικά επικοινωνεί με μια βάση δεδομένων. Τα συστήματα άμεσων μηνυμάτων εταιρικής κλάσης ενσωματώνουν IM bots για να παρέχουν έναν βολικό τρόπο στους εργαζόμενους να αναζητούν πληροφορίες. Ορισμένα IM bots μπορούν να συνδεθούν σε εξωτερικές βάσεις δεδομένων και να παρέχουν στον χρήστη ειδήσεις, αναφορές καιρού, οδηγίες οδήγησης, ώρες ταινιών, αποσπάσματα μετοχών και σχεδόν οποιοσδήποτε άλλες πληροφορίες μπορούν να βρεθούν σε μια βάση δεδομένων. Τα IM bots ονομάζονται επίσης διαδραστικοί παράγοντες (interactive agents). [1]

Τα chatbots χρησιμοποιούν διάλογο για να αποσπασουν τις πληροφορίες που χρειάζονται, για να φέρουν εις πέρας τα αιτήματα του χρήστη. Χωρίζονται σε διάφορες κατοιώριες, με σημαντικότερες την υγεία, τη ψηχαγωγία, την εκπαίδευση τις ειδήσεις και το e-commerce. [2] Τα περισσότερα είναι προσβάσιμα online μέσω chatting apps, ή μέσω virtual assistants.

1.1.2. Virtual Assistants

Οι voice assistants, ή virtual assistants είναι software agents που μπορούν να ερμηνεύσουν την ανθρώπινη ομιλία και να ανταποκριθούν μέσω συνθετικών φωνών. Η Siri της Apple, η Alexa της Amazon, η Cortana της Microsoft και η Google Assistant της Google είναι οι πιο δημοφιλείς virtual assistants και είναι ενσωματωμένες σε smartphones, υπολογιστές, ή εξειδικευμένα ηχεία στο σπίτι. Οι χρήστες μπορούν να υποβάλουν ερωτήσεις στους βοηθούς τους, μπορούν να ελέγχουν τις συσκευές οικιακού αυτοματισμού και την αναπαραγωγή πολυμέσων μέσω φωνής και να διαχειρίζονται άλλες βασικές εργασίες, όπως email, λίστες υποχρεώσεων και ημερολόγια με προφορικές εντολές. [3]

1.2. Turing Test

Το imitation game, το οποίο αποτελεί αυθεντική ονομασία του τότε διαδεδομένου Turing test, τέθηκε από τον Άγγλο μαθηματικό Alan Turing το 1950, ως η δοκιμασία της ικανότητας ενός μηχανήματος να επιδεικνύει ευφυή συμπεριφορά ισοδύναμη, ή πανομοιότυπη με αυτήν, ενός ανθρώπου. [4] Η εξάπλωση του Turing test κίνησε το ενδιαφέρον του Γερμανοαμερικανού computer scientist Joseph Weizenbaum, ο οποίος τέθηκε να δημιουργήσει το πρόγραμμα ELIZA για να αποδείξει την επιφανειακή επικοινωνία μεταξύ ανθρώπων και μηχανών. Το πρόγραμμα ELIZA πραγματοποιήθηκε την περίοδο 1964-1966 και προσομοίωνε συνομιλία χρησιμοποιώντας pattern matching και μια μεθοδολογία υποκατάστασης που έδινε στους χρήστες μια ψευδαίσθηση κατανόησης εκ μέρους του προγράμματος, παρόλο που δεν υπήρχε ενσωματωμένο framework για την κατασκευή του. Το προαναφερθέν θεωρείται ένα από τα πρώτα chatbots που έχουν δημιουργηθεί. Ο τρόπος κατασκευής του προγράμματος ELIZA ακολουθείτε μέχρι σήμερα για την κατασκευή chatbots. [5]

1.3. Artificial Intelligence

Η τεχνίτη νοημοσύνη (Artificial Intelligence), ή ως γνωστή από τη συντομογραφία της, AI, ορίζεται από την ικανότητα ενός ψηφιακού υπολογιστή ή ενός ελεγχόμενου από υπολογιστή ρομπότ να εκτελεί εργασίες που σχετίζονται συνήθως με ευφυή όντα. Υπάρχουν πολλές διαφορετικές μορφές μάθησης που εφαρμόζονται στην τεχνητή νοημοσύνη. Η απλούστερη είναι η εκμάθηση με δοκιμή και λάθος (trial and error). [6] Αυτού του είδους τεχνητή νοημοσύνη είναι ευρέως γνωστή ως μηχανική μάθηση (machine learning).

1.3.1. Machine Learning

Σε μεγάλες εταιρείες, όπως σε νοσοκομεία και αεροπορικούς οργανισμούς, οι μηχανικοί πληροφορικής σχεδιάζουν αρχιτεκτονικές για έξυπνα chatbots που χρησιμοποιούνται για να ανακαλύπτουν και να μοιράζονται τις γνώσεις και την εμπειρία στον οργανισμό πιο αποτελεσματικά και να μειώνουν σημαντικά τα λάθη στις απαντήσεις από γραφεία ειδικών υπηρεσιών, το οποίο καθιστάτε εφικτό από τη χρήση μηχανικής μάθησης. [7]

Η μηχανική μάθηση δίνει στο chatbot τη δυνατότητα να μαθαίνει από τις συνομιλίες του με τους χρήστες ούτως ώστε να μπορεί να παρουσιάσει ιδανικότερες απαντήσεις σε παρόμοιους διαλόγους. Για να επιτευχθεί αυτό, ένα αλγοριθμικό μοντέλο μηχανικής μάθησης είναι απαραίτητο και στη συνέχεια χρειάζονται αρκετά δεδομένα, συνήθως generated από χρήστες, καθώς και training το οποίο δίνει τη δυνατότητα στο μοντέλο να επεξεργαστεί τα δεδομένα, με σκοπό να μπορεί να χρησιμοποιήσει κάποια από αυτά σε μελλοντικούς διαλόγους.

1.4. Frameworks

Ένα software framework είναι μια πλατφόρμα για την ανάπτυξη εφαρμογών λογισμικού. Παρέχει τα θεμέλια στα οποία οι προγραμματιστές μπορούν να δημιουργήσουν προγράμματα για μια συγκεκριμένη πλατφόρμα. Για παράδειγμα, ένα framework μπορεί να περιλαμβάνει προκαθορισμένες κλάσεις και λειτουργίες που μπορούν να χρησιμοποιηθούν για την επεξεργασία εισόδου, τη διαχείριση συσκευών υλικού και την αλληλεπίδραση με λογισμικό συστήματος. [8]

Υπάρχουν διάφορα frameworks τα οποία χειρίζονται τη κατασκευή bots, που προσφέρουν διάφορα προνόμια στο προγραμματιστή για την ευκολότερη υλοποίηση του bot, καθώς και διάφορα μοντέλα τιμολόγησης. Κάποια από τα πιο δημοφιλή είναι τα εξής:

- Dialogflow (*Google*) [9]
- Watson Assistant (*IBM*) [10]
- botpress (*Botpress*) [11]
- Azure Bot Service (*Microsoft*) [12]

1.4.1. Azure Bot Service

Το Bot Framework, μαζί με το Azure Bot Service, παρέχει εργαλεία για τη δημιουργία, δοκιμή, ανάπτυξη και διαχείριση έξυπνων bots, όλα σε ένα μέρος. Το Bot Framework περιλαμβάνει ένα αρθρωτό και επεκτάσιμο SDK για την κατασκευή bots, καθώς και εργαλεία, πρότυπα και συναφείς υπηρεσίες AI. Με αυτό το framework, ο προγραμματιστής μπορεί να δημιουργήσουν bots που χρησιμοποιούν ομιλία, κατανοούν τη φυσική γλώσσα, χειρίζονται ερωτήσεις και απαντήσεις και πολλά άλλα. [13]

Διαθέτει SDK (Κιτ Ανάπτυξης Λογισμικού) ανοιχτού κώδικα (open source) για μεγαλύτερη ευελιξία και αξιοπιστία. [14]

Συνδυάζεται πλήρως με Azure Cognitive Services για να προσφέρει AI δυνατότητες στο chatbot καθιστώντας το εξυπνότερο. Τεχνολογίες όπως LUIS και QnA Maker έχουν χρησιμοποιηθεί και στα πλαίσια αυτής της διπλωματικής εργασίας για να δώσουν στο bot τη δυνατότητα να αντιλαμβάνεται φυσική γλώσσα και να απαντά ερωτήσεις οι οποίες ερωτώνται συχνά σε ένα bot. Παρόλα αυτά, τα Cognitive Services εμπεριέχουν και πολλές άλλες τεχνολογίες οι οποίες θα μπορούσαν να φανούν χρήσιμες για μελλοντική χρήση, οπότε η συμβατότητα τους με το bot είναι πάντα ευπρόσδεκτη.

Η ανάπτυξη ενός chatbot μέσω Azure Bot Service υποστηρίζεται από πολλές πλατφόρμες, τα λεγόμενα κανάλια. Συνδέοντας το bot με ένα κανάλι, δίνει τη δυνατότητα στους χρήστες του συγκεκριμένου καναλιού να συνομιλήσουν με το bot. Το ActivityCoachingBot είναι διαθέσιμο στο Microsoft Teams και στο Web Chat.

1.4.2. Clustering

Μια χρήσιμη εφαρμογή του Machine Learning είναι η δυνατότητα να ομαδοποιήσουμε οντότητες μεταξύ τους. Για παράδειγμα, αν θέλουμε να ομαδοποιήσουμε μουσική για να μπορούμε να την εντοπίσουμε με περισσότερη ευκολία. Η ομαδοποίηση οντοτήτων χωρίς επικέτες (labels), ονομάζεται clustering. Στη περίπτωση που δοθεί όνομα (label) σε κάθε ομάδα, τότε αυτή η ομαδοποίηση ονομάζεται classification. Στο classification χωρίζουμε τις οντότητες σε ήδη υπαρχόν ομάδες, καθώς στο clustering οι ομάδες δημιουργούνται με βάση τα χαρακτηριστικά των οντοτήτων που έχουμε στη διάθεσή μας. Για να επιτευχθεί το clustering παρατηρούμε ομοιότητες μεταξύ των οντοτήτων με συγκεκριμένες μετρικές, που ονομάζονται similarity measures. Τα χαρακτηριστικά των οντοτήτων ονομάζονται features και με βάση αυτά καθορίζεται το cluster. Μετά τη πραγματοποίηση του clustering, κάθε cluster λαμβάνει ένα μοναδικό αναγνωριστικό

το οποίο είναι το cluster ID του. Περνώντας τα πολλαπλά χαρακτηριστικά κάθε οντότητας μέσα από έναν clustering αλγόριθμο, μπορούμε να την αναθέσουμε σε ένα cluster στο οποίο θα υπάρχουν οντότητες με παρόμοια χαρακτηριστικά. [15]

1.4.3. Rules Engine

Ένα Rules Engine, ή ένα “Business Rules Engine” είναι ένα σύστημα λογισμικού που εκτελεί έναν ή περισσότερους κανόνες σε ένα runtime environment. Ένα rules engine επιτρέπει σε αυτές τις πολιτικές της εταιρείας και άλλες επιχειρησιακές αποφάσεις να καθορίζονται, να δοκιμάζονται, να εκτελούνται και να συντηρούνται ευκολότερα και γρηγορότερα. Τα rules engines συνήθως υποστηρίζουν κανόνες (rules), γεγονότα (facts), προτεραιότητα (score), αμοιβαίο αποκλεισμό (mutual exclusion), προϋποθέσεις (preconditions) και άλλες λειτουργίες. Συνήθως παρέχεται ως συστατικό ενός συστήματος διαχείρισης κανόνων το οποίο, μεταξύ άλλων λειτουργιών, παρέχει τις δυνατότητες: εγγραφής, ορισμού, ταξινόμησης και διαχείρισης όλων των κανόνων καθώς και επαλήθευση της συνέπειας των κανόνων.

Υπάρχουν διάφοροι τύποι rules engine. Αυτοί οι τύποι (γενικά) διαφέρουν ως προς τον τρόπο με τον οποίο οι κανόνες είναι προγραμματισμένοι. Η εφαρμογή των περισσότερων κανόνων που χρησιμοποιούνται από τις επιχειρήσεις είναι forward chaining, η οποία μπορεί να χωριστεί περαιτέρω σε δύο κατηγορίες:

- Η πρώτη τάξη επεξεργάζεται τους λεγόμενους κανόνες παραγωγής / συμπερασμάτων. Αυτοί οι τύποι κανόνων χρησιμοποιούνται για να αντιπροσωπεύουν συμπεριφορές του τύπου συνθήκης IF μετά από τη δράση.
- Στον δεύτερο τύπο το engine επεξεργάζεται τους λεγόμενους κανόνες αντίδρασης / συνθηκών δράσης (reaction / event condition action). Οι μηχανισμοί αντιδραστικού κανόνα εντοπίζουν και αντιδρούν σε εισερχόμενα συμβάντα και μοτίβα συμβάντων επεξεργασίας.

Η μεγαλύτερη διαφορά μεταξύ αυτών των τύπων είναι ότι οι μηχανές κανόνα παραγωγής εκτελούνται όταν ένας χρήστης ή μια εφαρμογή τους καλεί. Ένα reactive rules engine αντιδρά αυτόματα όταν συμβαίνουν γεγονότα. Πολλά δημοφιλείς rules engines έχουν δυνατότητες παραγωγής και αντίδρασης, αν και μπορεί να δίνουν έμφαση στη μία κατηγορία περισσότερο από την άλλη.

Μια τρίτη κατηγορία μηχανών κανόνων μπορεί να ονομαστεί ντετερμινιστική μηχανή. Για αυτούς τους κανόνες, οι μηχανές ενδέχεται να μην είναι ούτε forward, αλλά ούτε και backward chaining, και να χρησιμοποιούν domain specific language για την καλύτερη περιγραφή της πολιτικής. Αυτή η προσέγγιση είναι συχνά πιο εύκολη στην εφαρμογή και τη συντήρηση και παρέχει πλεονεκτήματα απόδοσης σε σχέση με τα forward chaining συστήματα.

Σε οποιαδήποτε εφαρμογή πληροφορικής, οι επιχειρηματικοί κανόνες αλλάζουν συχνότερα από τον υπόλοιπο κώδικα εφαρμογής. Οι μηχανές κανόνων ή inference engines είναι τα ενσωματωμένα στοιχεία λογισμικού που εκτελούν επιχειρηματικούς κανόνες που έχουν εξωτερικευτεί από τον κώδικα εφαρμογής ως μέρος μιας προσέγγισης επιχειρηματικών κανόνων. [16]

Το inference engine, πιο συγκεκριμένα στο χώρο του artificial intelligence είναι το σύστημα κανόνων που εφαρμόζει λογικούς κανόνες για να εξάγει πληροφορία από ένα knowledge base. Το σύστημα αποτελείται από το inference engine και το knowledge base. Το inference engine εφαρμόζει λογική στο knowledge base και εξάγει χρήσιμη πληροφορία. Αυτή η διαδικασία θα επαναληφθεί καθώς κάθε νέο fact στη βάση γνώσεων (knowledge base) θα μπορούσε να προκαλέσει πρόσθετους κανόνες στη μηχανή συμπερασμάτων. Οι inference engines λειτουργούν πρωτίστως σε έναν από τους δύο τρόπους είτε σε ειδικό κανόνα είτε σε γεγονότα: forward chaining και backward chaining. Το forward chaining ξεκινά με τα γνωστά γεγονότα και ισχυρίζεται νέα γεγονότα. Το backward chaining ξεκινά με στόχους και λειτουργεί προς τα πίσω για να καθορίσει ποια πραγματικά περιστατικά πρέπει να διεκδικήσει το rules engine ώστε να επιτευχθούν οι στόχοι. [17]

Το rules engine το οποίο χρησιμοποιήθηκε στα πλαίσια της διπλωματικής εργασίας είναι το NRules το οποίο είναι τύπου inference engine.

2. Ανάπτυξη conversational bot χρησιμοποιώντας Azure Bot Service

Το πρώτο βήμα για τη δημιουργία του chatbot επιτυγχάνεται μέσω του browser. Πλοηγούμαστε στο Azure Portal (portal.azure.com) και επιλέγουμε το Web App Bot resource. Για να το βρούμε ευκολότερα μπορούμε να το ψάξουμε στην μπάρα αναζήτησης. Στη συνέχεια συμπληρώνουμε όλα τα απαραίτητα στοιχεία για τη δημιουργία του bot μας. Σημαντικό να επιλέξουμε το Basic Bot σε C# για Bot template. Μπορούμε επίσης να δημιουργήσουμε ένα Resource Group στο οποίο θα οργανώσουμε όλα μας τα resources. Όταν η διαδικασία ολοκληρωθεί, δημιουργείτε το bot μας στο Azure, το οποίο έχει ήδη κάποιες βασικές λειτουργίες. Επίσης δημιουργείτε ένα καινούριο Authoring resource και App στο LUIS, το οποίο είναι συνδεδεμένο και πλήρως λειτουργικό με το bot που μόλις δημιουργήσαμε.

Από το Web App Bot που δημιουργήσαμε στο Azure μπορούμε να κατεβάσουμε το source code του bot μας, και να ξεκινήσουμε να το παραμετροποιούμε. Ακόμη εκεί θα βρούμε τα κανάλια στα οποία το bot μας μπορεί να λειτουργήσει.

Για την επεξεργασία του κώδικα θα χρειαστούμε μερικά εργαλεία. Αρχικά, το Visual Studio είναι ένα Integrated Development Environment (IDE) το οποίο μας παρέχει ένα περιβάλλον καθώς και compilers, για να γράφουμε C# και να προσθέσουμε το δικό μας κώδικα στο bot. Επιπρόσθετα χρειαζόμαστε το Bot Application Template.

Όταν κάνουμε build και run τον κώδικά μας μέσω του Visual Studio, εκκινείτε ένας server στον οποίο τρέχει το bot μας. Ο server αυτός χρησιμοποιεί το port 3978 του υπολογιστή μας. Για να επικοινωνήσουμε με το bot μας σε αυτό το port, χρειαζόμαστε τον Bot Framework Emulator (V4) τον οποίο μπορούμε να κατεβάσουμε από το GitHub.

2.1. LUIS

Το LUIS αναλύει τη φυσική γλώσσα που χρησιμοποιεί ο χρήστης και επιστρέφει στο bot τα intents και τα entities. Τα intents υποδεικνύουν τη πρόθεση του χρήστη. Για να προσθέσουμε κάποια πιθανή πρόθεση, πλοηγούμαστε στο LUIS portal (www.luis.ai) και επιλέγουμε το app που είναι συνδεδεμένο με το bot μας, στη συνέχεια πάμε στα Intents και πατάμε create για να δημιουργήσουμε ένα καινούριο intent, ή επιλέγουμε ένα από τα έτοιμα (prebuilt).


Intents

[+ Create](#) [+ Add prebuilt domain intent](#) [Rename](#) [Delete](#)

Σχήμα 1: Δημιουργία Intent

Μετάπειτα μπορούμε να προσθέσουμε φράσεις και προτάσεις στο intent μας για να βοηθήσουμε το μοντέλο να αντιληφθεί πως λειτουργεί το συγκεκριμένο intent.

Greet 

Machine learning features 

[+ Add feature](#)

Examples 

✓ Confirm all entities Move to Delete ...		View options Filter Reset
Example user input		Score
<input type="text" value="Type an example of what a user might say and hit Enter."/>		
hey !		0.974
hi !		0.950

Σχήμα 2: Καταχώρηση Intent


Με τον ίδιο τρόπο μπορούμε να δημιουργήσουμε Entities. Τα entities είναι μεταβλητές τις οποίες το LUIS μαθαίνει να αναγνωρίζει από τα λεγόμενα του χρήστη, όπως είναι τα ονόματα και οι ηλικίες. Μπορούν να δημιουργηθούν με τον ίδιο τρόπο όπως και τα intents.

Entities

[+ Create](#) [Add prebuilt entity](#) [Add prebuilt domain entity](#) [Rename](#) [Delete](#)

Σχήμα 3: Δημιουργία Entity

Όταν το bot έχει χρησιμοποιηθεί για ένα διάστημα, είτε μέσω emulator είτε σε published περιβάλλον, οι προτάσεις που γράφει ο χρήστης στο bot (utterances) μαζεύονται στο tab "Review Endpoint Utterances". Από εκεί μπορούμε να δούμε τι έχει σταλεί στο bot και να το κατατάξουμε στα ανάλογα intents και entities.

Review Endpoint Utterances 

Filter: [Greet](#) [Refresh](#)

[✓ Confirm all entity predictions](#) [Save](#) [Discard](#)

☐ Utterance

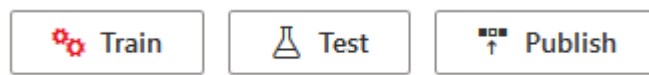
Predicted Intent

can i update my personal data

[EnterPersonalDetails \(0. ...\)](#)

Σχήμα 4: Review Endpoint Utterances

Τέλος για να μπορεί το bot να εκμεταλλευτεί τις αλλαγές που έχουμε κάνει, πρέπει να κάνουμε train και publish το LUIS μοντέλο μας.



Σχήμα 5: Train, Test και Publish το LUIS μοντέλο

Στο κώδικα του bot, θα πρέπει να ενημερωθούν τα αρχεία LuisModel.cs και LuisModel.json με τα καινούρια intents και entities για να είναι το καινούριο μοντέλο συμβατό με το bot.

2.2. Σύνδεση QnA Maker

Με το QnA Maker μπορούμε να δημιουργήσουμε βάσεις γνώσεων (knowledge bases), οι οποίες αποτελούνται από ερωτήσεις που θα μπορεί να απαντήσει το bot, μαζί με τις απαντήσεις τους.

Στο “Create a knowledge base” μπορούμε να φτιάξουμε το δικό μας knowledge base συμπληρώνοντας όλα τα στοιχεία της φόρμας. Στο πεδίο “Chit-chat” συμπληρώνουμε πιο είδος του chit-chat που επιθυμούμε να έχουμε στο bot μας, ανάλογα με το ύφος που θέλουμε το bot μας να έχει. Το Chit-chat module παρέχει 90 pairs από συχνές ερωτήσεις, μαζί με τις απαντήσεις τους, για να κάνει το bot μας να φαίνεται πιο συζητήσιμο.

Chit-chat

Give your bot the ability to answer thousands of small-talk questions in a voice that fits your brand. When you add chit-chat to your knowledge base by selecting a personality below, the questions and responses will be automatically added to your knowledge base, and you'll be able to edit them anytime you want. [Learn more about chit-chat](#).

- ☐ None
- ☐ Professional
- ☒ Friendly
- ☐ Witty
- ☐ Caring
- ☐ Enthusiastic

Σχήμα 6: Δημιουργία Chit-chat

Για να χρησιμοποιήσουμε το chit-chat από το bot, χρειάζεται να εγκαταστήσουμε το Microsoft.Bot.Builder.AI.QnA NuGet package και να συνδέσουμε τα credentials του καινούριου knowledge base μας με το bot.

2.3. Dialog

Η χρήση διαλόγων στο bot είναι άκρος ουσιώδης. Δίνουν στο bot τη δυνατότητα να κάνει συζήτηση με το χρήστη για να συλλέξει δεδομένα ή να αποδώσει πληροφορίες. Ο τρόπος που δουλεύει ο διάλογος στο bot framework, είναι με βήματα (WaterfallStep)

τα οποία εκτελούνται σειριακά χρησιμοποιώντας μοντέλο καταρράκτη (WaterfallDialog). Για την επικοινωνία με το χρήστη χρησιμοποιούνται prompts. Τα prompts, παρέχουν στο bot τη δυνατότητα να παρουσιάζει μηνύματα στο χρήστη, καθώς και σημαντικότερα, να λαμβάνει τα δεδομένα που ο χρήστης στέλνει στο bot.

Το κάθε prompt καθορίζει το τρόπο που ο χρήστης θα αλληλοεπιδρά μαζί του. Το πιο κοινό είδος prompt είναι το TextPrompt, το οποίο αφήνει το χρήστη να εισάγει κείμενο (text) στο bot. Για την κατασκευή του bot χρησιμοποιήθηκαν διάφορα prompts. Το NumberPrompt, το οποίο αφήνει το χρήστη να εισάγει αριθμούς, το ChoicePrompt το οποίο παρουσιάζει στο χρήστη επιλογές και τον αφήνει να επιλέξει μία από αυτές και το ConfirmPrompt το οποίο τον αφήνει αν απαντήσει θετικά ή αρνητικά σε μία ερώτηση. Στα πλαίσια της δημιουργίας αυτού του bot χρειάστηκε να φτιαχτεί και ένα custom prompt το οποίο παρουσιάζει adaptive cards στο χρήστη και του δίνει τη δυνατότητα να συμπληρώσει δεδομένα πάνω στη κάρτα η οποία χρησιμοποιείτε σαν κομμάτι διαλόγου. Αυτό παρέχει στο χρήστη ένα πιο ευχάριστο τρόπο εισαγωγής δεδομένων, πέρα από τον απλό διάλογο.

Κάποια από τα προαναφερθέντα prompts, όπως το ChoicePrompt και το ConfirmPrompt περιέχουν built-in validator ο οποίος επικυρώνει ότι ο χρήστης επέλεξε μια από τις διαθέσιμες επιλογές. Αν αυτό δε γίνει τότε το prompt του ξαναεμφανίζει την ερώτηση. Στη περίπτωση του TextPrompt, τέτοιος validator δεν υπάρχει, καθώς δεν είναι ξεκάθαρο ποια δεδομένα το prompt θεωρεί αποδεκτά. Σε αυτή τη περίπτωση χρειάζεται να δημιουργήσουμε το δικό μας validator, ο οποίος θα ελέγχει ότι τα δεδομένα που το συγκεκριμένο prompt χρειάζεται να συλλέξει, έχουν καταχωρηθεί με επιτυχία.

2.4. Σύνδεση bot με βάση δεδομένων (Azure Cosmos DB)

Αρχικά, είναι απαραίτητη η δημιουργία ενός Azure Cosmos DB resource, η οποία μπορεί να ολοκληρωθεί μέσω του Azure portal. Για το δεδομένο bot χρησιμοποιήθηκε το Core SQL API, το οποίο μας επιτρέπει να κάνουμε SQL queries στη βάση δεδομένων. Μόλις η δημιουργία της βάσης δεδομένων ολοκληρωθεί, μας παρέχεται η δυνατότητα να δημιουργήσουμε containers.

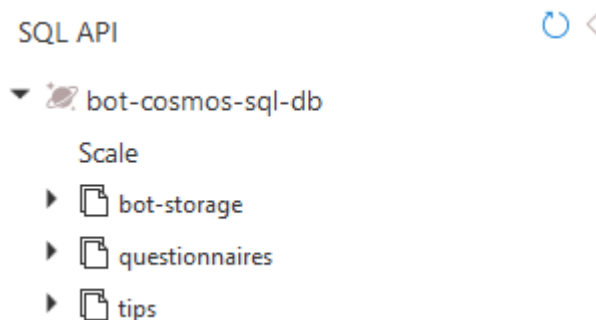
Το κάθε container λειτουργεί σαν ένα table σε μία συνηθισμένη βάση δεδομένων. Παρόλα αυτά, δεν χρειάζεται να αναθέσουμε τύπο δεδομένων στο κάθε container, καθώς μπορεί να δεχτεί objects οποιουδήποτε τύπου δεδομένων. Αυτό το καθιστά πολύ εύκολο στη λειτουργία του από το bot, καθώς μπορούμε να στείλουμε στη βάση ένα object τύπου PersonalDetails, με όλα τα προσωπικά δεδομένα του χρήστη και θα αποθηκευτεί κατευθείαν στη βάση. Η επικοινωνία της βάσης δεδομένων με το bot γίνεται εξολοκλήρου με τη χρήση JSON. Στη περίπτωση που θέλουμε να λάβουμε τα δεδομένα κάποιου χρήστη από τη βάση δεδομένων, τότε ζητάμε το object από τη βάση που ανήκει στο συγκεκριμένο ID και στη συνέχεια κάνουμε deserialize το JSON και cast στο object που θέλουμε να χρησιμοποιήσουμε.

Για να επιτευχθεί η σύνδεση του Cosmos DB με το bot χρειάζεται η δημιουργία συναρτήσεων οι οποίες συνδέονται με το Cosmos DB resource, χρησιμοποιώντας το endpoint και το κλειδί της βάσης.

Σε περίπτωση που χρειαζόμαστε να παραμετροποιήσουμε τον τρόπο που λαμβάνουμε τα δεδομένα από τη βάση, μπορούμε να χρησιμοποιήσουμε συνηθισμένα SQL queries για να λάβουμε τα δεδομένα. Για παράδειγμα, όταν θέλουμε να λάβουμε

τα δεδομένα για να κάνουμε το clustering, χρειαζόμαστε μόνο τα personality traits, αλλά τα χρειαζόμαστε από όλους τους χρήστες, δηλαδή από όλα τα records μέσα στο container. Αυτό μπορεί εύκολα να επιτευχθεί με το εξής SQL query: "SELECT c.document.Extraversion, c.document.Agreeableness, c.document.Conscientiousness, c.document.Neuroticism, c.document.Openness FROM c".

Οποιαδήποτε στιγμή επιθυμούμε, μπορούμε να δούμε τα δεδομένα που περιέχονται μέσα στη βάση δεδομένων από το Data Explorer στο Azure portal. Τα δεδομένα έχουν μορφή JSON.



Σχήμα 7: Βάση Δεδομένων μέσω του Data Explorer

Παρόλο που το Azure Cosmos DB παρέχει χαμηλό latency επικοινωνίας, οι χρόνοι τους οποίους ο χρήστης αναγκάζεται να περιμένει για να λάβει τα δεδομένα του μπορεί να προβούν ενοχλητικοί. Για ελαχιστοποίηση της καθυστέρησης κατά τη διάρκεια χρήσης του bot μπορούν να χρησιμοποιηθούν Tasks τα οποία έχουν ως λειτουργία την παραλαβή δεδομένων από τη βάση. Για παράδειγμα, για να εντοπίσουμε αν ένας χρήστης υπάρχει στη βάση δεδομένων χρειαζόμαστε το UserID. Οπότε αντί ο χρήστης να περιμένει την ανταπόκριση από τη βάση μόλις χρειαστεί να εισάγει δεδομένα, μπορούμε να ζητάμε τα δεδομένα από τη βάση, κατευθείαν μόλις λάβουμε το UserID. Με τη χρήση Tasks αυτή η λειτουργία, μπορεί να γίνει σε ένα άλλο process το οποίο ολοκληρώνεται στο background όσο ο χρήστης επικοινωνεί με το bot. Στη περίπτωση που τα δεδομένα δεν έχουν ακόμη φθάσει στο bot όταν ο χρήστης χρειάζεται να τα χρησιμοποιήσει, πρέπει να περιμένει εκεί μέχρι τα δεδομένα να φθάσουν. Αυτό επιτυγχάνεται με τη χρήση της await.

2.5. Clustering με ML.NET

Για να εφαρμόσουμε το clustering μέσω ML.NET, πρέπει να προσθέσουμε το ML.NET Model Builder στοιχείο στην εγκατάσταση του Visual Studio.

Τα στοιχεία από τα οποία καθορίζεται η επιλογή του cluster ονομάζονται features. Στην προκειμένη περίπτωση θέτουμε ως features τα πέντε personality traits (Extraversion, Agreeableness, Conscientiousness, Neuroticism, Openness). Για να διαχωρίσουμε τους χρήστες με βάση αυτά τα στοιχεία, χρειαζόμαστε ένα dataset. Το dataset μας θα είναι ένας πίνακας με αυτά τα πέντε personality traits από κάθε χρήστη τα οποία τα λαμβάνουμε με ένα SQL query. Τα υπόλοιπα στοιχεία κάθε χρήστη δε μας είναι χρήσιμα αυτή τη στιγμή, καθώς οι υπολογισμοί γίνονται με βάση τα features αποκλειστικά. Στη συνέχεια εφαρμόζουμε τον αλγόριθμο K-Means, υποδεικνύοντας τα features με σκοπό τη δημιουργία πέντε διαφορετικών clusters. Εκπαιδεύουμε (train) το dataset μας και έχουμε έτοιμο το clustering μοντέλο μας. Τέλος, αυτό που απομένει

είναι να προβλέψουμε (predict) το cluster στο οποίο ανήκει ο παρόν χρήστης και αυτό γίνεται με το prediction engine του ML.NET. Αφού έχουμε το cluster του χρήστη, το καταχωρούμε στην αντίστοιχη μεταβλητή στα στοιχεία του χρήστη.

2.6. Εφαρμογή Rules Engine με NRules

Η χρήση του Rules Engine στο bot είναι πολύ σημαντική καθώς καθορίζει τη σωστή επιλογή των tips ανά χρήστη. Για την επίτευξη αυτού του έργου το rules engine που χρησιμοποιούμε είναι το NRules. Για τη χρήση του NRules στο bot, χρειαζόμαστε το ομώνυμο NuGet package.

Για τη δημιουργία ενός καινούριου κανόνα, δημιουργούμε ένα class το οποίο κάνει inherit το class με όνομα "Rule". Μέσα στο class, ορίζουμε όλες τις μεταβλητές που θα χρειαστούμε και τους αναθέτουμε τιμή "default", καθώς η τιμή τους θα αποδοθεί από την εσωτερική μνήμη του rules engine. Στη συνέχεια το When() statement καθορίζει τον κανόνα, καθώς το Then(), καθορίζει τι συμβαίνει όταν αυτός ο κανόνας τηρείτε.

Αφότου ολοκληρώσουμε όλους τους κανόνες, πρέπει να υποδείξουμε ένα σημείο στον κώδικα στο οποίο τρέχει το rules engine. Στη δική μας περίπτωση, αυτό το σημείο είναι όταν πρέπει να εμφανιστεί ένα tip στο χρήστη. Για να τρέξουμε το rules engine δημιουργούμε ένα RuleRepository στο οποίο φορτώνουμε τους κανόνες που δημιουργήσαμε. Δεν είναι απαραίτητο να δοθούν όλοι οι κανόνες, αφότου δοθεί ο πρώτος κανόνας, το NRules αναλαμβάνει να προσθέσει και τους υπόλοιπους κανόνες από μόνο του. Στη συνέχεια κάνουμε compile τους κανόνες και δημιουργούμε ένα session του rules engine. Το session έχει τη δική του μνήμη στην οποία αποθηκεύει όλες τις μεταβλητές που χρησιμοποιούν οι κανόνες. Οπότε εμείς πρέπει να κάνουμε insert όλα τα δεδομένα μας μέσα στο session. Μόλις όλα είναι έτοιμα, κάνουμε fire το session και το NRules μας επιστρέφει όλα τα tips που είναι κατάλληλα για τον συγκεκριμένο χρήστη.

2.7. Ενεργοποίηση proactive μηνυμάτων

Για να εμφανίζονται τα tips στους χρήστες γίνεται λειτουργία των proactive messages, τα οποία μας δίνουν τη δυνατότητα να γίνονται notified όλοι οι χρήστες όταν ένα συγκεκριμένο endpoint κληθεί. Για τη περιγραφή αυτής της λειτουργικότητας χρειάζεται η δημιουργία ενός controller.

Ο NotifyController ενημερώνεται κάθε φορά που σταλθεί get request στο endpoint του bot. Είναι υπεύθυνος να κάνει respond σε αυτό το get request με μία html σελίδα, η οποία ενημερώνει ότι το endpoint κλήθηκε επιτυχώς, καθώς και να καλεί μία συνάρτηση η οποία θα παρουσιάζει στο χρήστη ένα tip. Για λόγους ομαλότητας, γίνεται πρώτα ένας έλεγχος ότι έχει ήδη ανατεθεί cluster στο συγκεκριμένο χρήστη, πριν ξεκινήσουν να του παρουσιάζονται tips. Ο τρόπος με τον οποίο ο controller εντοπίζει τον κάθε χρήστη που έχει επικοινωνήσει με το bot, είναι κρατώντας το conversation reference του κάθε χρήστη μέσα σε ένα dictionary. Οπότε μόλις το endpoint κληθεί, το μήνυμα αποστέλλεται στους κατόχους όλων των conversation references.

Τα conversation references συλλέγονται από τη συνάρτηση OnMessageActivityAsync η οποία καλείτε για κάθε μήνυμα που αποστέλλει κάποιος χρήστης στο bot.

Η ενεργοποίηση αυτών των μηνυμάτων επί καθημερινής βάσης αναλαμβάνεται από ένα Azure Function. Αυτό το function περιέχει ένα trigger και τον κώδικα τον οποίο χρειάζεται για να ολοκληρώσει ένα get request στο endpoint του bot. Το trigger

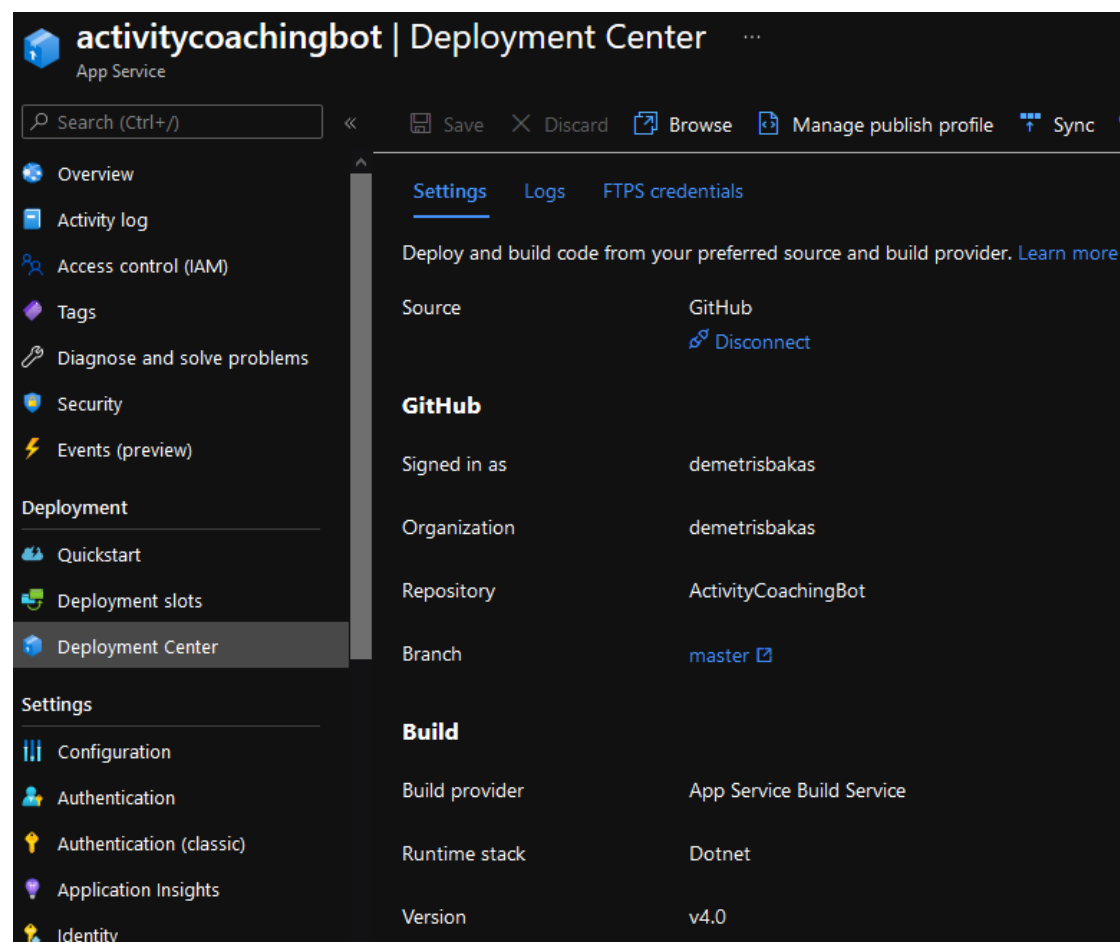
ενεργοποιείτε καθημερινά στις 18:00 UTC μέσω ενός cron expression. Το trigger με τη σειρά του ενεργοποιεί το κώδικα του function, το οποίο κάνει get request στο endpoint του bot, όπου ο controller αναλαμβάνει να παρουσιάσει το tip σε κάθε χρήση.

2.8. Publication

Το bot μπορεί πολύ εύκολα να γίνει publish κατευθείαν από το Visual Studio. Παρόλα αυτά, για την ευκολότερη ανάπτυξή του, αυτή η διαδικασία έχει αυτοματοποιηθεί.

Για την ανάπτυξη του bot έχει χρησιμοποιηθεί git, ως μορφή source control. Το GitHub χρησιμοποιήθηκε ως η πλατφόρμα ή οποία έκανε host το source code όλης της εφαρμογής. Υλοποιήθηκαν δύο git branches, το “development” branch, για να αποθηκεύεται η ανάπτυξη της εφαρμογής, και το “master” branch στο οποίο εμφανίζονταν όλα τα versions του bot που γίνονταν released. Η επικοινωνία του GitHub με το local git πραγματοποιήθηκε εξολοκλήρου μέσω Sourcetree.

Στο Deployment Center του App Service του bot, έγινε σύνδεση μεταξύ του master branch του ActivityCoachingBot repository στο GitHub, με το App Service του bot στο Azure. Αυτό δίνει τη δυνατότητα, κάθε φορά που γίνεται commit κάτι καινούριο στο master branch στο GitHub, αυτός ο κώδικας να χρησιμοποιείται για να γίνει build και deploy το καινούριο version της εφαρμογής.



Σχήμα 8: Σύνδεση Deployment Center με GitHub

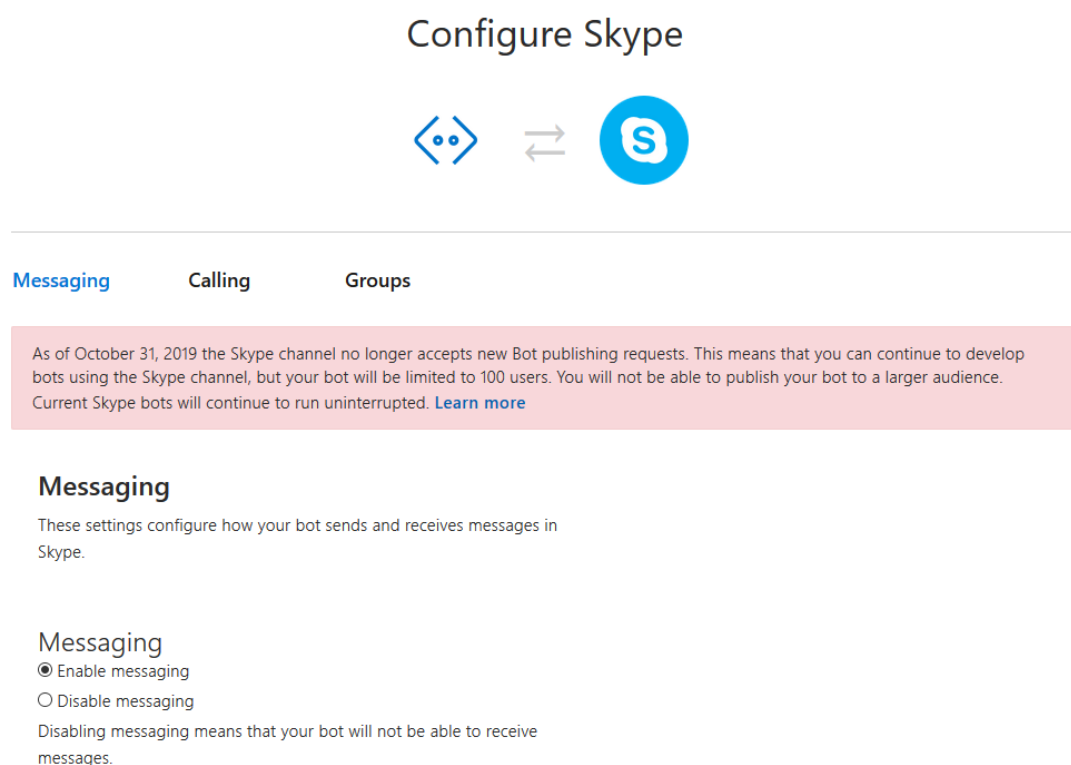
2.9. Επιλογή Καναλιών

Τα κανάλια στα οποία η εφαρμογή είναι διαθέσιμη για χρήση, είναι τα Microsoft Teams και Web Chat. Η επιλογή καναλιών γίνεται από το tab “Channels” του Web App Bot που έχουμε δημιουργήσει στο Azure Portal. Με πολύ εύκολο τρόπο μπορεί να επεκταθεί η χρήση του bot και σε περεταίρω κανάλια.

Για το δοθέν παράδειγμα θα γίνει επέκταση στο κανάλι “Skype”. Όπως φαίνεται και στο σχήμα, το Skype έχει κάποιους περιορισμούς με τη χρήση chatbots και για αυτό το λόγο δεν είναι ένα από τα κανάλια στα οποία το bot είναι επίσημα διαθέσιμο. Παρόλα αυτά ο τρόπος με τον οποίο γίνεται η επέκταση καναλιού είναι ο ίδιος με τα ήδη υπάρχοντα κανάλια.

Αξίζει να σημειωθεί ότι ο τρόπος με τον οποίο εμφανίζονται τα μηνύματα και οι κάρτες εξαρτώνται σε ένα βαθμό και από το κανάλι. Οπότε η εμφάνιση του bot πιθανό να διαφέρει σε κάποιο από τα καινούρια κανάλια που προστίθενται.







Για την εισαγωγή καινούριου καναλιού, πλοηγούμαστε στο Web App Bot μας. Από το Channels tab επιλέγουμε το Skype και το παραμετροποιούμε όπως επιθυμούμε. Μόλις ολοκληρώσουμε το wizard, το bot μας είναι διαθέσιμο και στο Skype.



Σχήμα 9: Επιλογή Καινούριου Καναλιού

Για να το χρησιμοποιήσουμε πατάμε πάνω στο κανάλι που θέλουμε, στην αρχική σελίδα του Channels tab.

Connect to channels

Name	Health	Published	Actions
 Microsoft Teams	Running	--	Edit 
 Skype	Running	--	Edit 
 Web Chat	Running	--	Edit 

[Get bot embed codes](#)

Σχήμα 10 : Κανάλια που Χρησιμοποιεί το bot

Υποστηρίζονται και κανάλια τα οποία χρησιμοποιούν φωνή, όπως η Alexa της Amazon. Το bot έχει υλοποιηθεί με τέτοιο τρόπο ούτως ώστε να υποστηρίζονται τα φωνητικά μηνύματα, αλλά η χρησιμότητα των adaptive cards είναι περιορισμένη.

Το κανάλι Web Chat, μας επιτρέπει να ενθέσουμε το bot σε κάποιο website, ή απλά να το χρησιμοποιήσουμε μέσω του browser μας. Αυτό προϋποθέτει ότι ο χρήστης θα δίνει όλα τα στοιχεία που υπό κανονικές συνθήκες προσφέρονται από το κανάλι, το οποίο τα αντλεί από το λογαριασμό του χρήστη. Παρόλα αυτά το bot διαθέτει αυτή τη λειτουργικότητα και ρωτά το χρήστη για όλες τις πληροφορίες που χρειάζεται, ανεξάρτητος καναλιού.

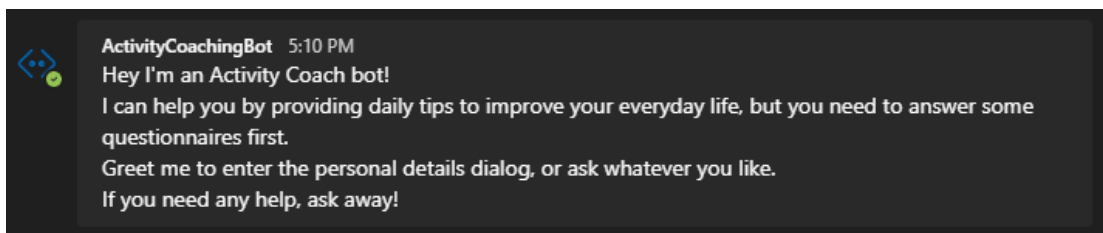
3. Εγχειρίδιο Χρήσης /

Εγκατάστασης

Σε αυτή την ενότητα περιγράφεται η χρήση του bot μέσω της πλατφόρμας Microsoft Teams. Ωστόσο, το Microsoft Teams καθιστά μόνο μία από τις πλατφόρμες τις οποίες το bot υποστηρίζει, αλλά ο τρόπος λειτουργίας του είναι όμοιος και στις υπόλοιπες πλατφόρμες.

Για τη λειτουργία σε αυτή τη πλατφόρμα ο χρήστης χρειάζεται να έχει ένα λογαριασμό στο Microsoft Teams, καθώς και πρόσβαση σε αυτό, είτε μέσω browser, είτε μέσω client για υπολογιστή ή κινητό.

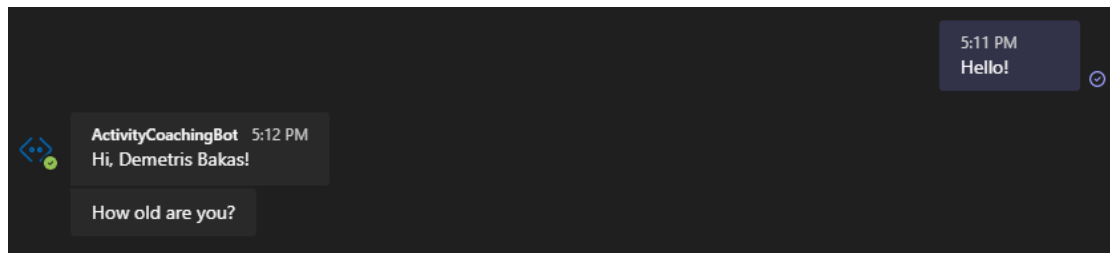
Πιθανό η συζήτηση να μη ξεκινήσει αμέσως και να χρειαστεί ο χρήστης να στείλει ένα αρχικό μήνυμα στο bot για να εκκινήσει η λειτουργία του. Στη συνέχεια το bot καλωσορίζει το χρήστη με το αρχικό μήνυμα και ο χρήστης καλείται να επικοινωνήσει με φυσική γλώσσα για να πει στο bot πως θέλει να συνεχίσει.



Σχήμα 11: Καλωσοριστικό Μήνυμα

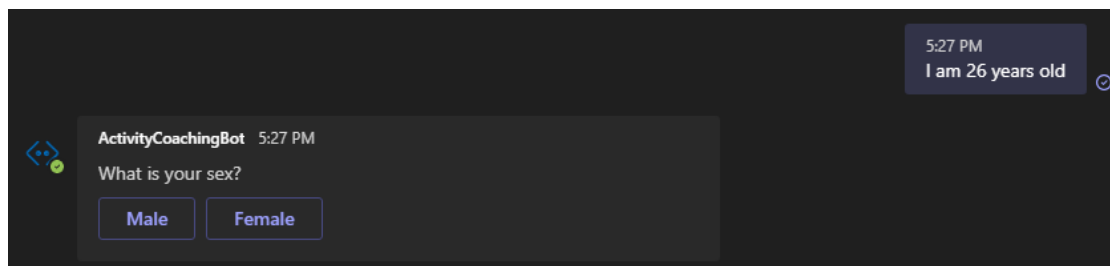
Από αυτό το σημείο υπάρχουν πολλαπλοί τρόποι να συνεχίσει ο χρήστης. Μπορεί να χαιρετήσει το bot, το οποίο θα του ανοίξει το διάλογο για να συμπληρώσει τα προσωπικά του στοιχεία. Μπορεί επίσης να ζητήσει κατευθείαν στο bot να συμπληρώσει τα προσωπικά του στοιχεία, ή να απαντήσει ένα ερωτηματολόγιο, όπου θα μεταφερθεί ακριβώς στον ίδιο διάλογο. Επίσης υπάρχει η δυνατότητα ο χρήστης να ζητήσει να ανεβάσει tips ή ερωτηματολόγια στη βάση δεδομένων. Κάτι το οποίο για να πραγματοποιηθεί, ο χρήστης χρειάζεται να έχει στη κατοχή του έναν κωδικό διαχειριστή.

Για την ώρα θα στείλουμε ένα καλωσοριστικό μήνυμα. Στη συνέχεια το bot χαιρετά το χρήστη με το όνομά του και αρχίζει το διάλογο για να συλλέξει τα προσωπικά στοιχεία του χριστή ξεκινώντας από την ηλικία, δεδομένου ότι ο χρήστης δεν είχε δώσει την ηλικία του στο προηγούμενο μήνυμα.



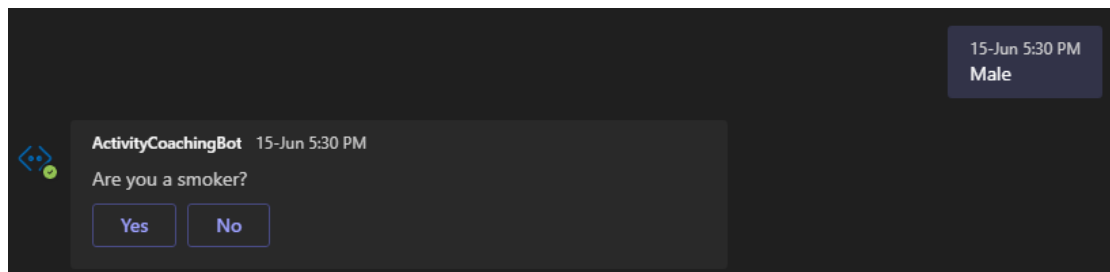
Σχήμα 12: Ερώτηση Ηλικίας

Στην επόμενη ερώτηση το bot ρωτά το χρήστη για το φύλο του και δικαιούται να πατήσει σε μία από της ακόλουθες επιλογές. Αν ο χρήστης επιθυμεί να γράψει την απάντησή του στο bot, δικαιούται, φτάνει να είναι μία από τις υπάρχοντες απαντήσεις.



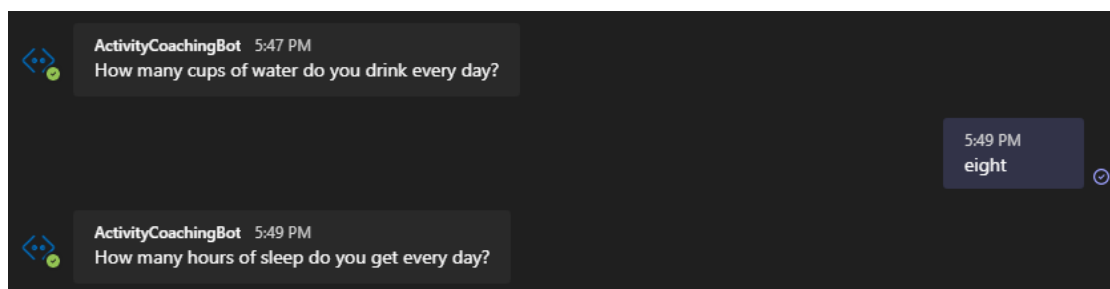
Σχήμα 13: Ερώτηση Φύλου

Μετάπειτα το bot ρωτά το χρήστη αν είναι καπνιστής και ο χρήστης καλείτε να επιλέξει μία από τις δύο επιλογές.



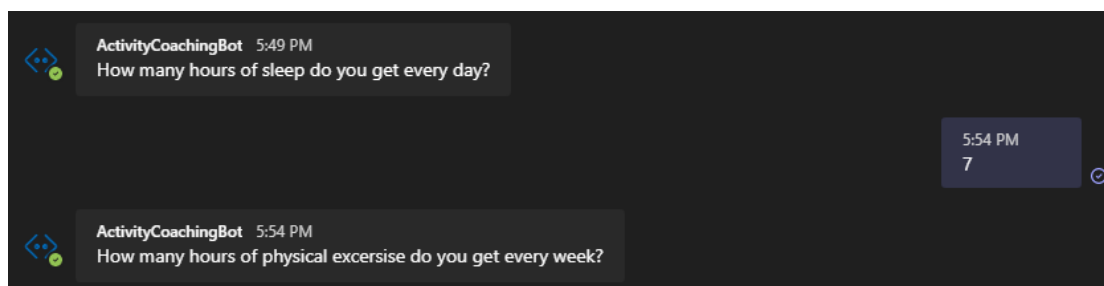
Σχήμα 14: Ερώτηση Καπνιστή

Αφότου ο χρήστης απαντήσει, το bot ρωτά τον χρήστη πόσα ποτήρια νερό καταναλώνει την ημέρα. Η ερώτηση μπορεί να απαντηθεί είτε με νούμερα, είτε με λέξεις.



Σχήμα 15: Ερώτηση Κατανάλωσης Νερού και Ύπνου

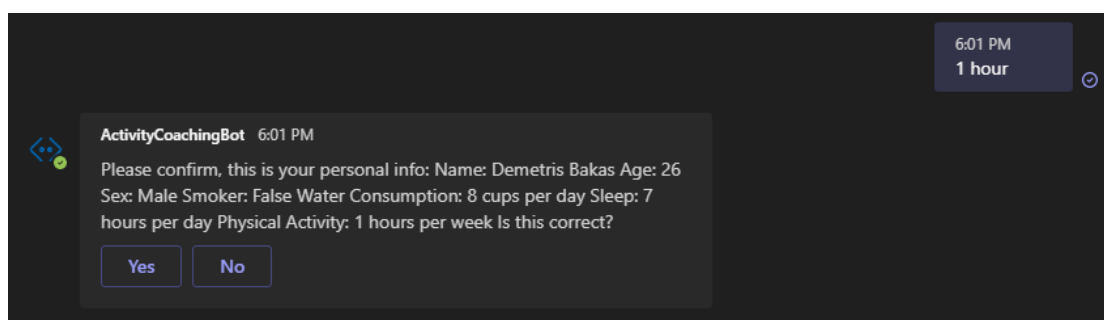
Η επόμενη ερώτηση είναι ο αριθμός ωρών που ο χρήστης κοιμάται την ημέρα και μπορεί να απαντηθεί με τον ίδιο τρόπο όπως και η προηγούμενη ερώτηση.



Σχήμα 16: Ερώτηση Φυσικής Άσκησης

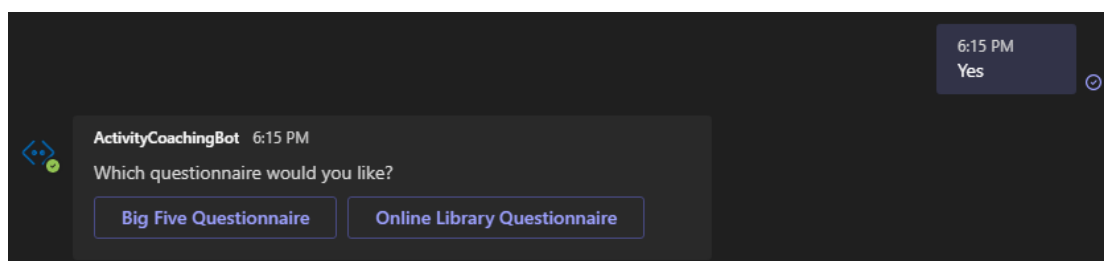
Η τελευταία ερώτηση για τα προσωπικά δεδομένα του χρήστη έχει να κάνει με ώρες που δαπανά ο χρήστης για φυσική άσκηση τη βδομάδα και μπορεί να απαντηθεί με τον ίδιο τρόπο όπως και οι δύο τελευταίες ερωτήσεις.

Στη συνέχεια όλα τα δεδομένα παρουσιάζονται στο χρήστη και ερωτάτε αν είναι ορθά. Αν απαντήσει αρνητικά, τότε η διαδικασία συλλογής δεδομένων ξεκινά ξανά από την αρχή. Αν ο χρήστης απαντήσει θετικά τότε προχωρά στην επιλογή ερωτηματολογίου.



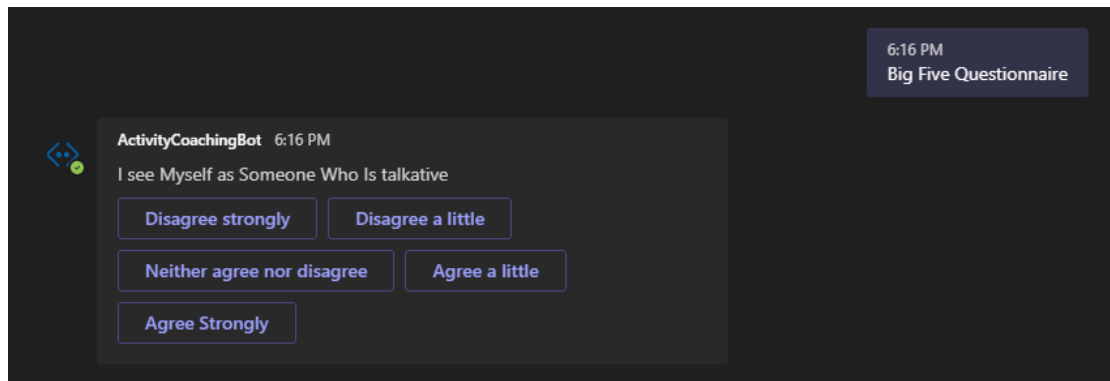
Σχήμα 17: Επικύρωση Στοιχείων

Αφού τα προσωπικά στοιχεία έχουν συμπληρωθεί, ο χρήστης μπορεί να φτάσει στο βήμα επιλογής ερωτηματολογίου ζητώντας από το bot να απαντήσει ερωτηματολόγια, πέρα από απλά χαιρετώντας το bot. Σε αυτό το βήμα καλείτε να επιλέξει πιο ερωτηματολόγιο θέλει να απαντήσει.



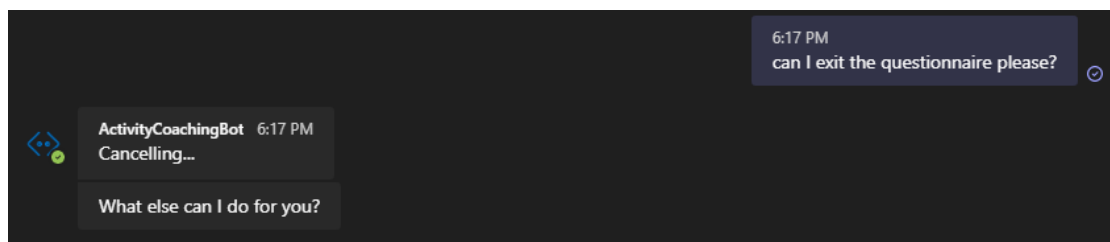
Σχήμα 18: Επιλογή Ερωτηματολογίου

Κάθε ερώτηση του κάθε ερωτηματολογίου εμφανίζεται στο χρήστη με πέντε πιθανές απαντήσεις. Ο χρήστης απλά επιλέγει την απάντησή του, ή τη γράφει ως κείμενο στο bot.



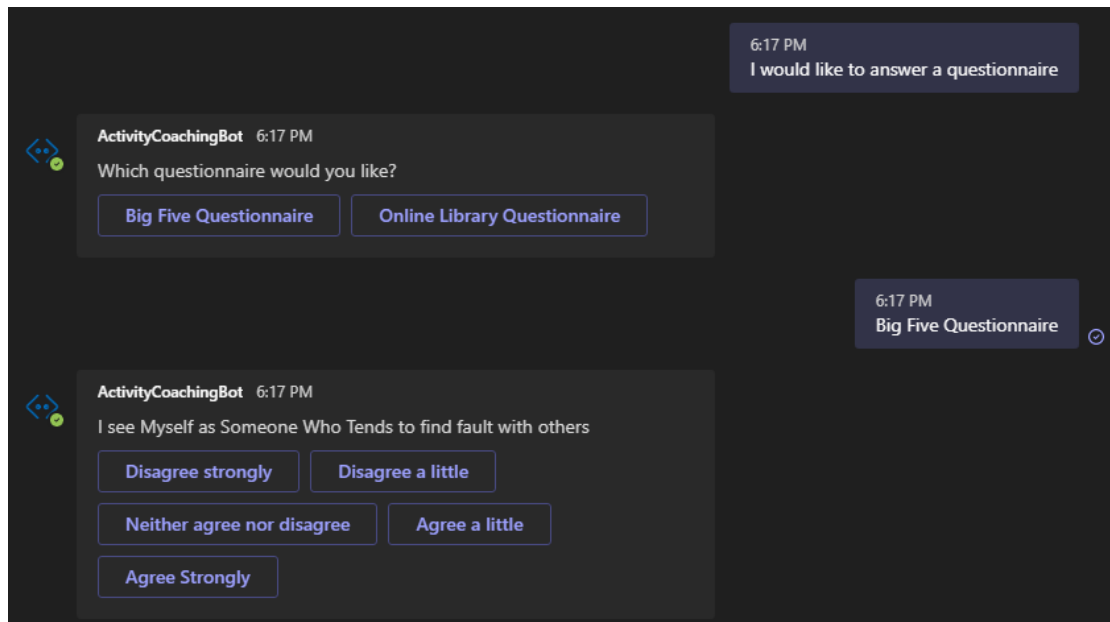
Σχήμα 19: Παρουσίαση Ερωτήσεων

Ανά πάσα στιγμή ο χρήστης μπορεί να ζητήσει από το bot να εξέλθει από τον παρόν διάλογο, όποιος και αν είναι αυτός. Στη παρακάτω εικόνα παρουσιάζεται η επιθυμία του χρήστη να διακόψει την απάντηση του ερωτηματολογίου. Στη περίπτωση που το ερωτηματολόγιο, ή οποιαδήποτε συλλογή δεδομένων διακοπεί, όταν ο χρήστης επανέλθει στο συγκεκριμένο διάλογο, θα επαναφερθεί στο σημείο που είχε μείνει προηγουμένως.



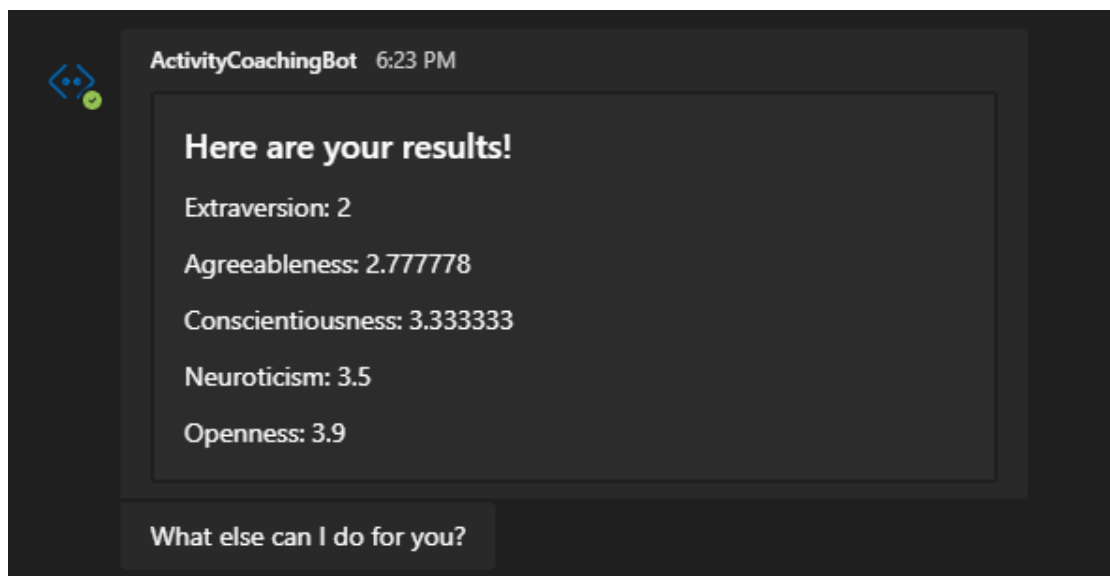
Σχήμα 20: Έξοδος Διαλόγου

Από τον αρχικό διάλογο, ο χρήστης, μπορεί να ζητήσει στο bot να απαντήσει κάποιο ερωτηματολόγιο. Επιλέγοντας το ερωτηματολόγιο που άφησε στη μέση, θα συνεχίσει από την ερώτηση που είχε μείνει.



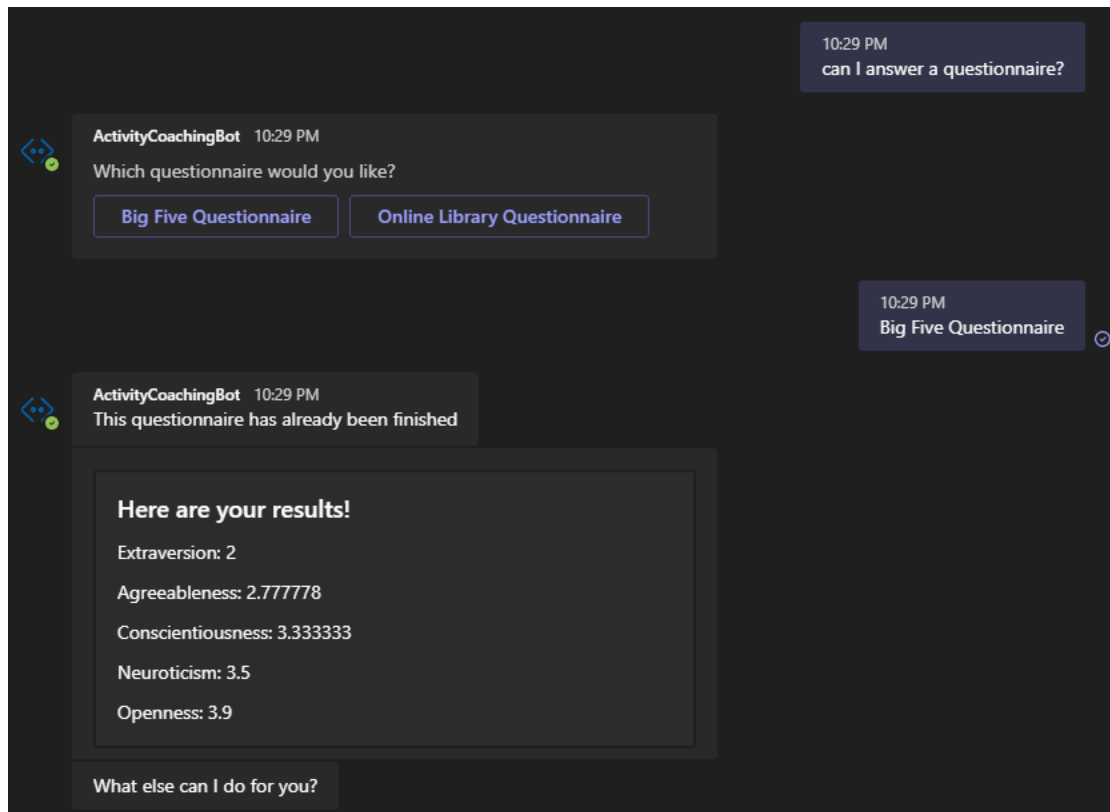
Σχήμα 21: Απάντηση Ερωτηματολογίων

Μόλις η απάντηση κάποιου ερωτηματολογίου ολοκληρωθεί, τα personality trait scores παρουσιάζονται στο χρήστη και στη συνέχεια το bot επιστρέφει στον αρχικό διάλογο.



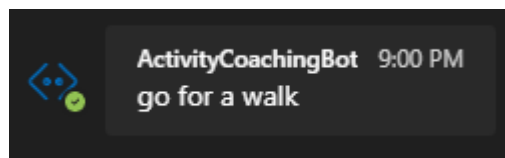
Σχήμα 22: Αποτελέσματα Personality Traits

Στη περίπτωση που ο χρήστης επιλέξει να απαντήσει ερωτηματολόγιο το οποίο έχει ήδη απαντηθεί το bot θα ενημερώσει το χρήστη ότι έχει ολοκληρώσει ήδη το ερωτηματολόγιο και θα του παρουσιάσει τα personality trait scores του.



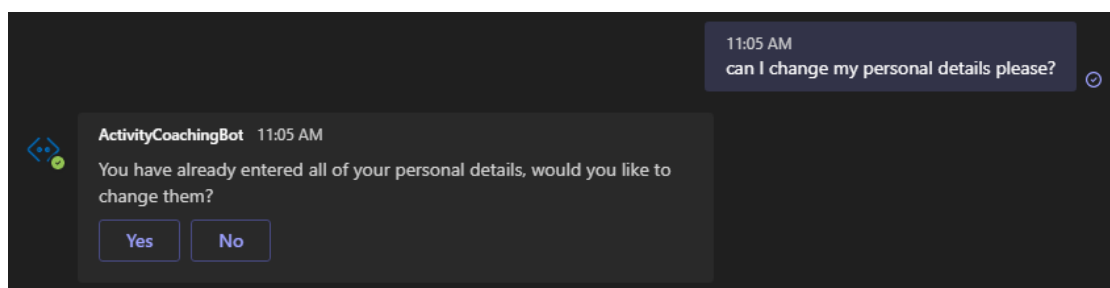
Σχήμα 23: Ολοκληρωμένο Ερωτηματολόγιο

Αφότου ο χρήστης ολοκληρώσει τουλάχιστον ένα ερωτηματολόγιο, θα ξεκινήσει να λαμβάνει εξατομικευμένα tips με βάση τις απαντήσεις, τις ασχολίες και το χαρακτήρα του, που είναι η βασική λειτουργία του bot. Όσα περισσότερα ερωτηματολόγια απαντήσει ο χρήστης, τόσο πιο ακριβής γίνονται τα tips που προσφέρει το bot.



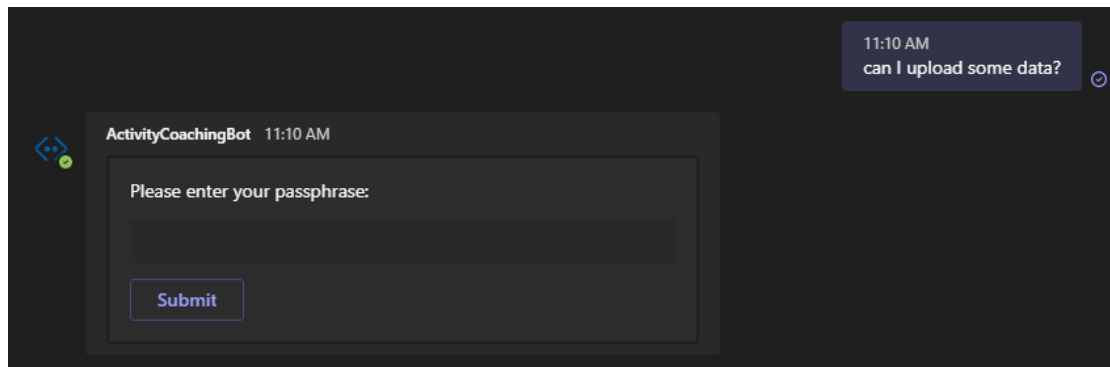
Σχήμα 24: Tip

Στη περίπτωση που ο χρήστης επιθυμεί να αλλάξει τα προσωπικά του δεδομένα, μπορεί να το ζητήσει από το bot, και μετά να απαντήσει θετικά στην ερώτηση επιβεβαίωσης.



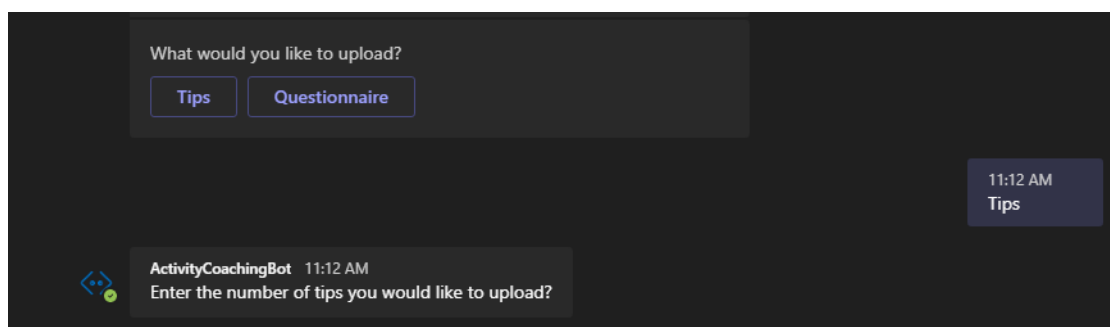
Σχήμα 25: Αλλαγή Προσωπικών Δεδομένων

Αν ο χρήστης θέλει να ανεβάσει ερωτηματολόγια ή tips στη βάση δεδομένων του bot μπορεί να το ζητήσει. Το bot θα του παρουσιάσει μία κάρτα στην οποία πρέπει να συμπληρώσει τον κωδικό διαχειριστή.



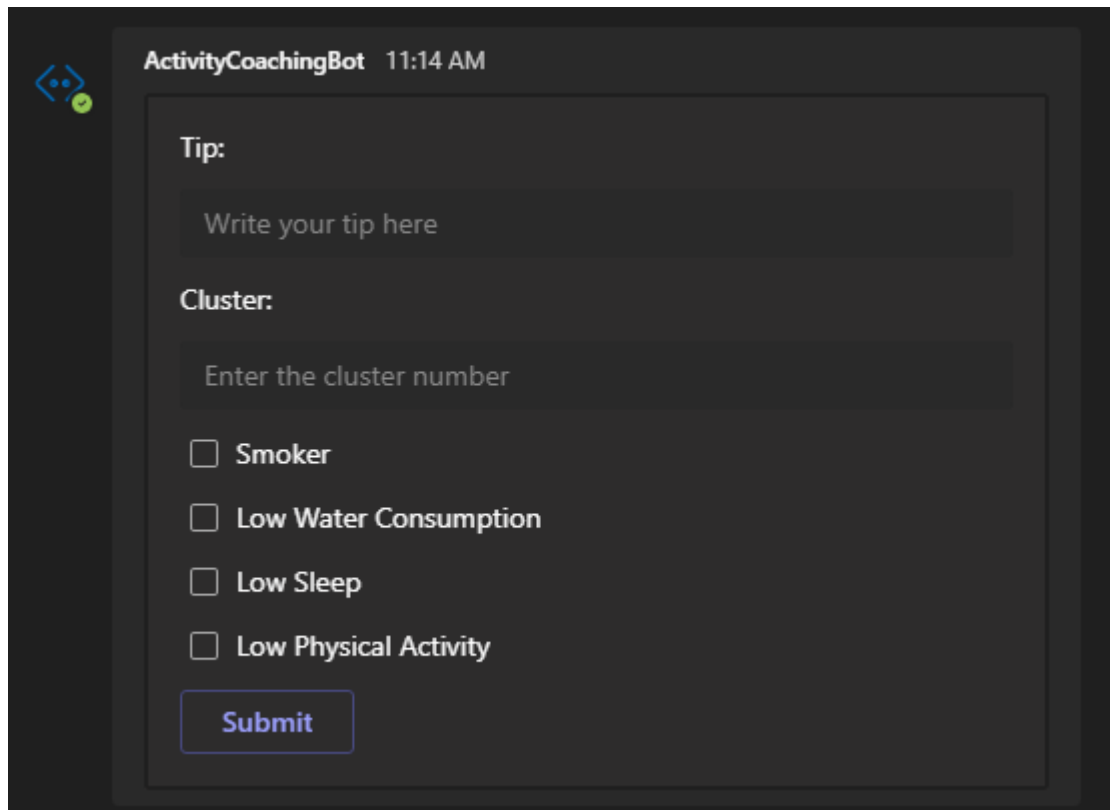
Σχήμα 26: Εισαγωγή Κωδικού Διαχειριστή

Μόλις ο κωδικός συμπληρωθεί επιτυχώς, το bot ρωτά το χρήστη αν επιθυμεί να ανεβάσει tips ή ερωτηματολόγιο. Αν ο χρήστης επιλέξει τα tips, τότε το bot τον ρωτά για τον αναμενόμενο αριθμό των tips που θέλει να ανεβάσει.



Σχήμα 27: Αριθμός Tips προς Αποστολή στη Βάση Δεδομένων

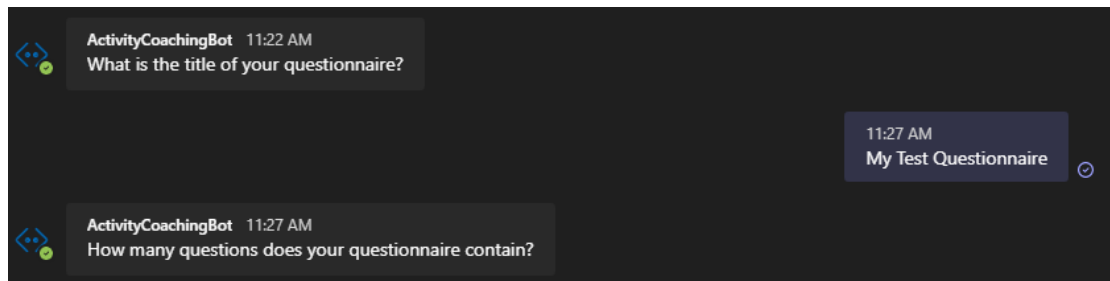
Όταν ο χρήστης απαντήσει του παρουσιάζονται κάρτες στις οποίες θα συμπληρώσει τα tips, η κάθε κάρτα αντιστοιχεί σε ένα tip. Όλα τα πεδία στη κάρτα είναι προαιρετικά, εκτός από το πεδίο "Tip". Μόλις όλες οι κάρτες συμπληρωθούν, το bot στέλνει αυτόματα τα tips στη βάση δεδομένων και επιστρέφει το χρήστη στον αρχικό διάλογο.



The screenshot shows a chat window for 'ActivityCoachingBot' at 11:14 AM. The interface is dark-themed. It contains a 'Tip:' section with a text input field labeled 'Write your tip here'. Below this is a 'Cluster:' section with a text input field labeled 'Enter the cluster number'. Underneath are four checkboxes: 'Smoker', 'Low Water Consumption', 'Low Sleep', and 'Low Physical Activity'. At the bottom of the form is a blue 'Submit' button.

Σχήμα 28: Κάρτα Εισαγωγής Tip

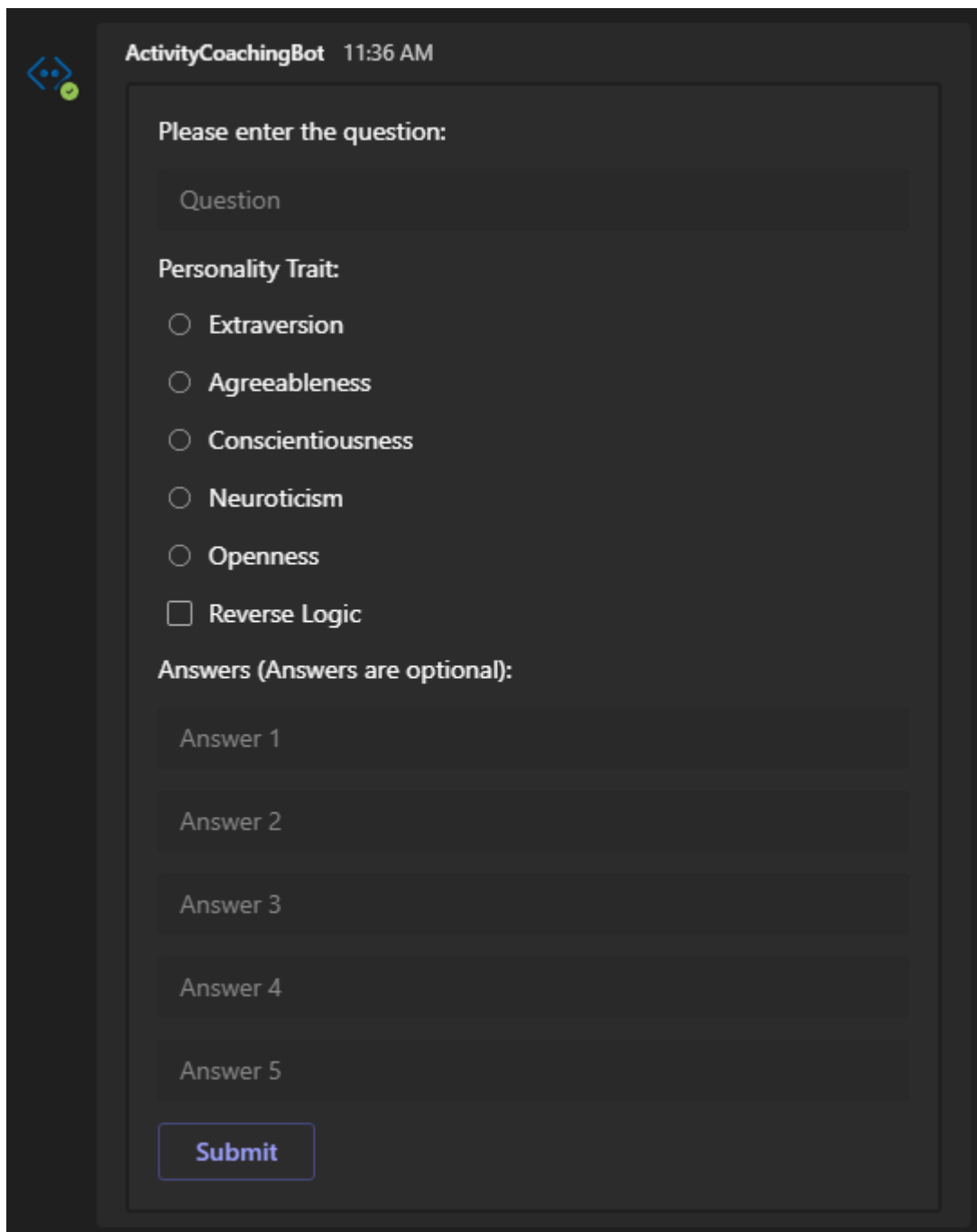
Αν ο χρήστης επιλέξει να ανεβάσει ερωτηματολόγιο, το bot ζητά το όνομα του ερωτηματολογίου. Μόλις ο χρήστης το συμπληρώσει, ερωτάτε για τον αριθμό των ερωτήσεων που περιέχονται στο ερωτηματολόγιο το οποίο επιθυμεί να ανεβάσει.



The screenshot shows a chat window for 'ActivityCoachingBot' with two messages. The first message, at 11:22 AM, asks 'What is the title of your questionnaire?'. The user's response, at 11:27 AM, is 'My Test Questionnaire'. The second message, also at 11:27 AM, asks 'How many questions does your questionnaire contain?'.

Σχήμα 29: Αριθμός Ερωτήσεων στο Ερωτηματολόγιο

Όταν ο χρήστης δώσει τον αριθμό των ερωτήσεων, εμφανίζονται κάρτες, μία για κάθε ερώτηση όπως και στα tips. Η κάθε κάρτα περιέχει τη πλήρη περιγραφή μίας ερώτησης. Τα πεδία "Answers" μπορούν να μείνουν κενά, με αποτέλεσμα η συγκεκριμένη ερώτηση να λαμβάνει τις default απαντήσεις. Όταν ο χρήστης συμπληρώσει όλες τις κάρτες, το bot ανεβάζει αυτόματα το ερωτηματολόγιο στη βάση δεδομένων και επιστρέφει το χρήστη στον αρχικό διάλογο.

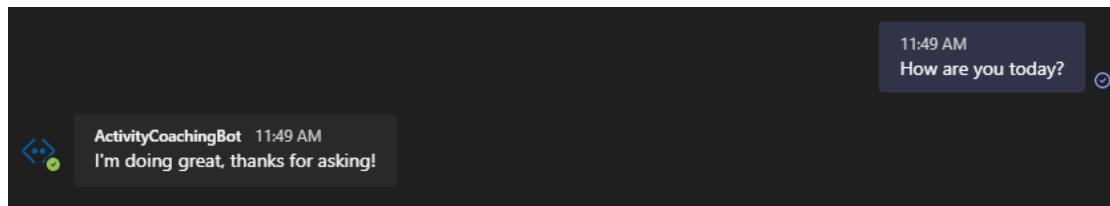


The image shows a dark-themed chat interface for 'ActivityCoachingBot' at 11:36 AM. The bot's icon is a blue hexagon with a green checkmark. The main content area contains a form with the following sections:

- Please enter the question:** A text input field with the placeholder 'Question'.
- Personality Trait:** A list of five radio button options: 'Extraversion', 'Agreeableness', 'Conscientiousness', 'Neuroticism', and 'Openness'. Below these is a checkbox labeled 'Reverse Logic'.
- Answers (Answers are optional):** Five text input fields labeled 'Answer 1' through 'Answer 5'.
- Submit:** A blue button with the text 'Submit'.

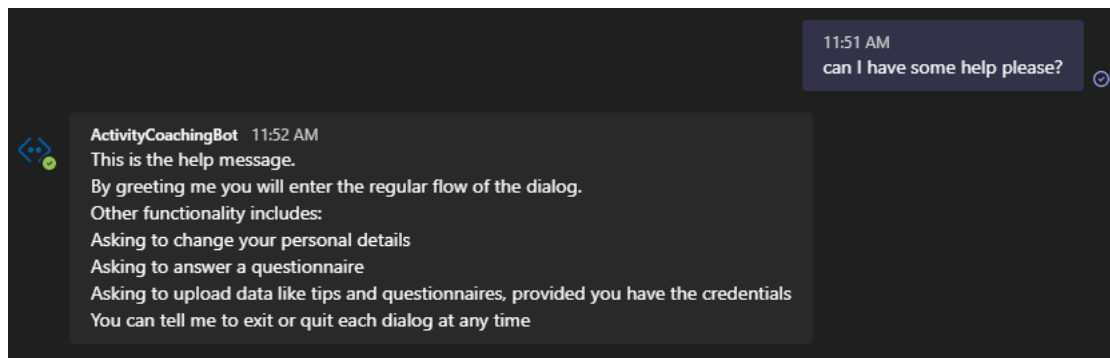
Σχήμα 30: Κάρτα Εισαγωγής Ερώτησης

Καθώς ο χρήστης βρίσκεται στο βασικό διάλογο, παρέχεται και η δυνατότητα να ρωτήσει κάποιες βασικές ερωτήσεις chit-chat το bot. Αυτές οι ερωτήσεις δεν αποσκοπούν σε κάποια συγκεκριμένη λειτουργία του bot. Απλά παρέχονται ούτως ώστε να βοηθούν το χρήστη να νιώθει πιο οικία μιλώντας στο bot.



Σχήμα 31: Ερώτηση Ρουτίνας (Chit-chat)

Τέλος, οποιαδήποτε στιγμή ο χρήστης χρειαστεί κάποια βοήθεια, ανεξάρτητα σε ποιο διάλογο βρίσκετε, μπορεί να ζητήσει βοήθεια από το bot, το οποίο κατευθείαν θα του παρουσιάσει το μήνυμα βοήθειας.



Σχήμα 32: Μήνυμα Βοήθειας

4. Αρχιτεκτονική / Υλοποίηση

4.1. Αρχιτεκτονική

Για την υλοποίηση του chatbot χρησιμοποιήθηκε Azure Bot Service και Microsoft Bot Framework. Αυτές οι υπηρεσίες παρέχουν εργαλεία για την δημιουργία, δοκιμή, ανάπτυξη και διαχείριση του chatbot. Το Bot Framework έχει ένα open source SDK (Software Development Kit) το οποίο είναι αρθρωτό και επεκτάσιμο δίνοντας πρόσβαση σε συσχετισμένες υπηρεσίες AI που βελτιώνουν την επικοινωνία με τον χρήστη. Η σύνταξη του bot έχει πραγματοποιηθεί εξολοκλήρου χρησιμοποιώντας τη γλώσσα προγραμματισμού C#.

Ο χρήστης επικοινωνεί κατευθείαν με το chatbot. Αυτό με τη σειρά του επικοινωνεί με το LUIS (Language Understanding) για να επιβεβαιώσει τις προθέσεις του χρήστη. Το LUIS χρησιμοποιεί ένα machine learning μοντέλο, εκπαιδευμένο με βάση τους διαλόγους του με χρήστες, για να αντιληφθεί προθέσεις και πληροφορίες που δίνει ο χρήστης, χρησιμοποιώντας φυσική γλώσσα.

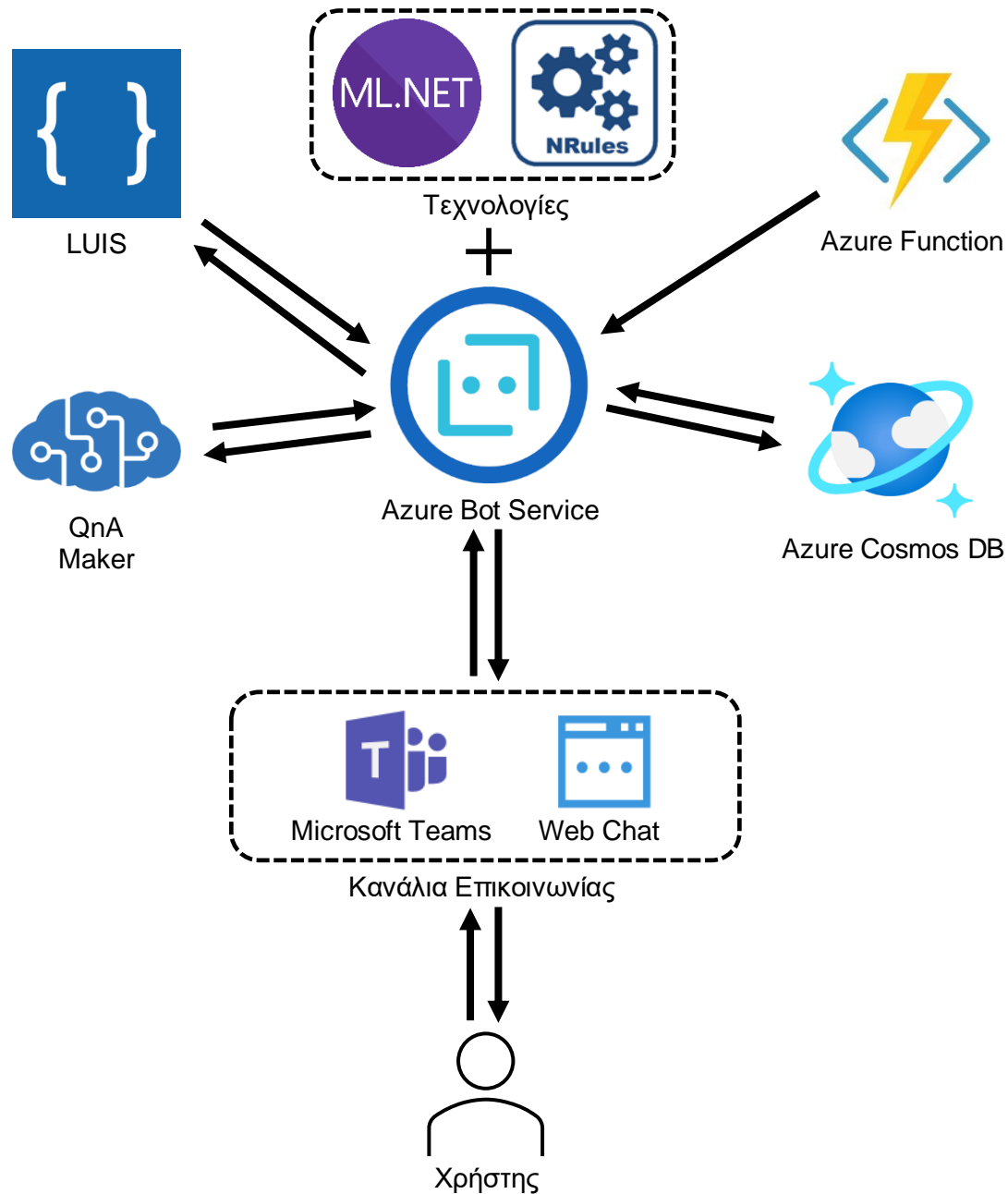
Αν οι προθέσεις δεν προδιαγράφονται σε αυτά που καλύπτει το chatbot, τότε καλείτε το QnA Maker service, για να δώσει μια πιθανή απάντηση στο χρήστη με στόχο να συνεχιστεί ο διάλογος, χρησιμοποιώντας το Chit-chat module. Το bot επίσης επικοινωνεί με μια βάση δεδομένων (Azure Cosmos DB) στην οποία αποθηκεύονται τα στοιχεία και οι απαντήσεις του χρήστη, τα questionnaires, καθώς και τα ανάλογα tips που δίνονται στο χρήστη.

Κάθε φορά που ο χρήστης απαντά πλήρως ένα ερωτηματολόγιο, διεξάγεται clustering βασισμένο στον αλγόριθμο K-means με τη χρήση ML.NET, το οποίο χωρίζει τους χρήστες σε πέντε ομάδες (clusters) ανάλογα με τους χαρακτήρες τους (τα big 5 personality traits) και κατατάσσει τον δεδομένο χρήστη σε μία από αυτές τις ομάδες (clusters).

Μία φορά τη μέρα ενεργοποιείτε ένα Azure Function το οποίο στέλνει στον κάθε χρήστη ένα μήνυμα που περιέχει το ημερήσιο tip του. Αφότου ενεργοποιηθεί το Azure Function, το tip επιλέγεται ξεχωριστά για κάθε χρήση με βάση το χαρακτήρα του, δηλαδή το cluster στο οποίο βρίσκετε και το τρόπο ζωής του, όπως είναι οι ώρες ύπνου και η φυσική άσκηση. Αυτή η επιλογή γίνεται με τη χρήση του rules engine NRules, το οποίο χρησιμοποιεί τον αλγόριθμο Rete για την γρήγορη επιλογή tips τα οποία είναι καταλληλότερα για το κάθε χρήστη.

Στη περίπτωση που ο χρήστης θέλει να ανεβάσει το δικό του ερωτηματολόγιο ή τα δικά του tips στη βάση δεδομένων μπορεί να το ζητήσει από το bot. Στη συνέχεια το bot απαιτεί από το χρήστη το κωδικό που αποδεικνύει ότι έχει δικαίωμα να ανεβάσει δεδομένα στη βάση δεδομένων. Αφού δοθεί ο κωδικός, το bot παρουσιάζει στο χρήστη κάρτες (Adaptive Cards) για να συμπληρωθούν είτε με ερωτήσεις, είτε με tips και στη συνέχεια τα κάνει upload στο Cosmos DB.

Όποτε επιθυμεί ο χρήστης μπορεί να ζητήσει βοήθεια από το bot ανεξάρτητα αν βρίσκετε μέσα σε κάποιο διάλογο. Όταν ζητηθεί βοήθεια το bot παρουσιάζει στο χρήστη κάποιους βασικούς τρόπους αλληλεπίδρασης. Επίσης οποιαδήποτε στιγμή ο χρήστης επιθυμεί, μπορεί να ζητήσει να αποχωρήσει από το συγκεκριμένο διάλογο. Όταν αποφασίσει να εκκινήσει εκείνο το διάλογο ξανά, θα συνεχίσει από το σημείο στο οποίο είχε μείνει τη προηγούμενη φορά.



Σχήμα 33: Σχεδιάγραμμα Λειτουργίας Τεχνολογιών

4.1.1. LUIS

Πιο συγκεκριμένα η υπηρεσία LUIS (Language Understanding Intelligence Service) που είναι μέρος των Azure Cognitive Services παρέχει τη δυνατότητα κατανόησης της φυσικής (ανθρώπινης) γλώσσας μέσω ενός μοντέλου Machine Learning. [18] Η

επικοινωνία επιτυγχάνετε χρησιμοποιώντας αιτήματα σε JSON format. Το chatbot στέλνει αυτό που είπε ο χρήστης (utterance) στην υπηρεσία LUIS η οποία το χωρίζει σε πιθανά intents και entities που μπορούν αν αφομοιωθούν από το bot για τη συνέχεια του διαλόγου. Το intent δείχνει πια είναι η πρόθεση του χρήστη στη συγκεκριμένη πρόταση. Τα entities λειτουργούν σαν μεταβλητές οι οποίες επηρεάζουν αυτό που στοχεύει να επιτύχει το bot.

```
query: "Hello there, you can call me Mark"
prediction:
  topIntent: "Greet"
  intents:
    Greet:
      score: 0.9992619
    None:
      score: 0.00109690067
    Cancel:
      score: 0.000118958087
  entities:
    personName:
      0: "Mark"
    $instance:
      personName:
        0:
          type: "builtin.personName"
          text: "Mark"
          startIndex: 29
          length: 4
          modelTypeId: 2
          modelType: "Prebuilt Entity Extractor"
      recognitionSources:
        0: "model"
```

Σχήμα 34: LUIS Response τύπου JSON

Για παράδειγμα, αυτό επιστρέφει το LUIS service όταν ο χρήστης πει “Hello there, you can call me Mark”. Την πρόθεση του χρήστη τη βλέπουμε από το topIntent το οποίο είναι “Greet” με ποσοστό επιτυχίας της εικασίας 0,9992619 από το 1. Στα entities βλέπουμε άλλες χρήσιμες μεταβλητές που περιείχε η πρόταση. Η μοναδική μεταβλητή στο πιο πάνω παράδειγμα είναι το personName που απεικονίζει το όνομα του χρήστη, το οποίο είναι “Mark”.

4.1.2. QnA Maker

Το QnA Maker είναι μια cloud-based υπηρεσία API η οποία είναι μέρος των Azure Cognitive Services, που επιτρέπει τη δημιουργία ενός επιπέδου συνομιλίας

ερωτήσεων-απαντήσεων πάνω από τα υπάρχοντα δεδομένα. Δίνει τη δυνατότητα δημιουργίας βάσεων γνώσεων, με δυνατότητα εξαγωγής ερωτήσεων και απαντήσεων για ενσωμάτωση με bot. Το Chit-chat είναι το τι απέγινε το NuGet package: Personality Chat. Δίνει τη δυνατότητα ενσωμάτωσης μικρών συνομιλιών σε bot για να μπορεί να απαντά σε συχνά χρησιμοποιούμενες ερωτήσεις χρηστών. Είναι πλέον διαθέσιμο μέσω του QnA Maker και μπορεί να χρησιμοποιείται από το bot, ακριβώς όπως οποιαδήποτε άλλη βάση γνώσεων (knowledge base). Η επικοινωνία με γίνεται με τη χρήση JSON όπως και με το LUIS. [19]

Στη περίπτωση του Activity Coaching Bot δε χρησιμοποιείτε κάποιο knowledge base πέρα από το Chit-chat. Το Chit-chat module έχει friendly χαρακτήρα και είναι υπεύθυνο να απαντά ερωτήσεις τις οποίες οι χρήστες συνηθίζουν να ρωτάνε σε chatbots, ή γενικότερα να χρησιμοποιούν σε ένα διάλογο (small talk). Η κάθε ερώτηση αποτελείται από πολλαπλά utterances, τα οποία αντιστοιχούν στην ίδια απάντηση. Ο λόγος που συμπεριλαμβάνετε στο bot είναι καθαρά για να κάνει το διάλογο μεταξύ του χρήστη και του bot να μοιάζει πιο φυσικός και να δίνει τη ψευδαίσθηση ενός “έξυπνου” bot στο χρήστη.

4.1.3. Azure Cosmos DB

Το Azure Cosmos DB είναι βάση δεδομένων, πλήρως διαχειριζόμενη σε NoSQL. Διαθέτει ταχείς χρόνους απόκρισης, enterprise grade security και availability. Χρησιμοποιεί και αυτό JSON για την επικοινωνία του με το bot και δίνει τη δυνατότητα αποθήκευσης αντικειμένων (objects) κατευθείαν στη βάση. Για την παρούσα υλοποίηση χρησιμοποιείτε SQL schema. [20]

Στη βάση δεδομένων αποθηκεύονται σε ένα container (table) τα προσωπικά δεδομένα των χρηστών μαζί με τις απαντήσεις και το cluster στο οποίο βρίσκεται ο χρήστης. Ένα άλλο container περιέχει όλα τα ερωτηματολόγια (questionnaires) με όλες τις ερωτήσεις και απαντήσεις τους. Η δυνατότητα custom απαντήσεων είναι προαιρετική, οποιαδήποτε ερώτηση δεν εμπεριέχει δικές τις απαντήσεις, συμπληρώνετε με τις default απαντήσεις από το bot κατά τη διάρκεια του runtime. Τέλος υπάρχει άλλο ένα container που περιέχει όλα τα tips που δίδονται στους χρήστες με το αντίστοιχο cluster τους και όλα τα προσωπικά χαρακτηριστικά που πιθανόν να επηρεάζουν. Όλα τα δεδομένα (PersonalDetails, Questionnaires, Tips) είναι αντικείμενα τα οποία εφαρμόζουν ακριβώς σε classes που υπάρχουν στο bot για την ευκολότερη χρήση τους. Τα PersonalDetails έχουν ως unique key το unique user ID που δίνετε από το bot σε κάθε χρήστη και χρησιμοποιείτε στη βάση δεδομένων ως το αναγνωριστικό για τον συγκεκριμένο χρήστη. Τα Questionnaires και Tips δε χρειάζονται αναγνωριστικό, καθώς φορτώνονται όλα μαζί στο bot, οπότε απλά χρησιμοποιείτε το όνομα του κάθε ερωτηματολογίου και το περιεχόμενο του κάθε tip, σε string. Όλα τα δεδομένα αποθηκεύονται δυναμικά και δεν υπάρχει όριο στα tips, τις ερωτήσεις ή τα ερωτηματολόγια που θα χρησιμοποιηθούν από το bot.

Το παρακάτω σχήμα δείχνει τα τρία containers της βάσης δεδομένων μαζί με τα πεδία που περιέχουν, αναλυτικά. Το κάθε container περιέχει objects στο πεδίο document, τον οποίων τα fields καταγράφονται στο σχήμα.

bot-storage	
id	(Primary Key) string (UserID)
document	PersonalDetails
UserID	string
Name	string
Age	int?

Sex	string
Smoker	bool?
WaterConsumption	int?
Sleep	int?
PhysicalActivity	int?
Extraversion	float?
Agreeableness	float?
Conscientiousness	float?
Neuroticism	float?
Openness	float?
Cluster	uint?
QuestionnaireAnswers	IDictionary<string, int>

Σχήμα 35: bot-storage Container

tips	
id	(Primary Key) string (TipMessage)
document	Tip
TipMessage	int?
Cluster	string
Smoker	bool?
LowSleep	bool?
LowWaterConsumption	bool?
LowPhysicalActivity	bool?

Σχήμα 36: tips Container

questionnaires	
id	(Primary Key) string (Name)
document	KeyValuePair<string, List<QuestionTopFive>>
Key: Name	string
Value: Questions	List<QuestionTopFive>
Question	string
Answers	List<Choice>
personalityTrait	enum PersonalityTrait
reverseLogic	bool

Σχήμα 37: questionnaires Container

4.1.4. ML.NET

Το ML.Net είναι ένα open source και cross-platform framework, που δημιουργήθηκε από τη Microsoft, το οποίο χρησιμοποιεί Machine Learning για να δώσει στον developer τη δυνατότητα να χειρίζεται εύκολα τα δεδομένα με δική του βούληση. Χρησιμοποιώντας τη διαθέσιμη δημιουργία μοντέλου, τα δεδομένα μπορούν να μετατραπούν σε prediction άμεσα. [21]

Αυτό το framework χρησιμοποιήθηκε για να υλοποιηθεί το clustering το οποίο είναι υπεύθυνο να διαχωρίζει τους χρήστες σε ξεχωριστές ομάδες (clusters) ανάλογα με τα χαρακτηριστικά (personality traits) τους. Ο διαχωρισμός γίνεται με τον αλγόριθμο K-means.

Τα στοιχεία τα οποία χρησιμοποιούνται για το clustering ονομάζονται features. Τα "Extraversion", "Agreeableness", "Conscientiousness", "Neuroticism", "Openness" καθιστούν ένα feature το καθένα. Ο διαμοιρασμός γίνεται σε πέντε clusters. Παρόλο που σε αυτή τη περίπτωση οι clusters έχουν τον ίδιο αριθμό με τα features, αυτό δε σημαίνει ότι ο κάθε cluster αντιπροσωπεύει ένα feature. Ο κάθε cluster αντιπροσωπεύει ένα τύπο ανθρώπινου χαρακτήρα (συνδυασμό features) στον οποίο διάφοροι χαρακτήρες με κοινά στοιχεία μπορούν να κατηγοριοποιηθούν.

K-means

Ο αλγόριθμος K-means για clustering είναι ένας επαναληπτικός αλγόριθμος που χωρίζει το σύνολο δεδομένων σύμφωνα με τα χαρακτηριστικά τους σε αριθμό K προκαθορισμένων, μη επικαλυπτόμενων, διακριτών clusters. Κάνει τα σημεία δεδομένων των inter clusters όσο το δυνατόν πιο παρόμοια και προσπαθεί επίσης να διατηρήσει τα clusters όσο το δυνατόν περισσότερο. Εκχωρεί τα σημεία δεδομένων σε ένα cluster, εάν το άθροισμα της τετραγωνικής απόστασης μεταξύ του κέντρου του cluster (cluster's centroid) και των σημείων δεδομένων είναι τουλάχιστον στο σημείο όπου το κέντρο του cluster είναι ο αριθμητικός μέσος όρος των σημείων δεδομένων που βρίσκονται στο cluster. Μια λιγότερη διακύμανση του cluster οδηγεί σε παρόμοια ή ομοιογενή σημεία δεδομένων εντός του cluster. [22]

4.1.5. NRules

Το NRules είναι ένα rules engine σχεδιασμένο για .NET και χρησιμοποιεί τον αλγόριθμο αντιστοίχισης Rete. Οι κανόνες γράφονται σε C# με χρήση internal DSL. Ανήκει στη κατηγορία των inference engines το οποίο του δίνει τη δυνατότητα να εκτελεί τους κανόνες χωρίς να ακολουθεί κάποια συγκεκριμένη, προκαθορισμένη σειρά. Υπολογίζει ποιοι κανόνες πρέπει να ενεργοποιηθούν βάσει των γεγονότων (facts) που του έχουν δοθεί και στη συνέχεια, τους εκτελεί σύμφωνα με έναν conflict resolution αλγόριθμο.

Αποτελείται από πολλά στοιχεία, και έχει σχεδιαστεί με τέτοιο τρόπο ώστε περισσότερα στοιχεία και εργαλεία να μπορούν να τοποθετούνται πάνω του, αυξάνοντας τη λειτουργικότητά του και επεκτείνοντας το πεδίο εφαρμογής του. Οι κανόνες είναι το κεντρικό κομμάτι του engine και υπάρχουν σε πολλαπλές μορφές, συγκεκριμένα στο NRules έχουν μορφή internal DSL που χρησιμοποιεί fluent API. Το internal DSL μεταφράζεται σε canonical form με στόχο την υποστήριξη περεταίρω γλωσσών κανόνων. Με τη χρήση του αλγόριθμου Rete το NRules προσπαθεί να βρει αποτελεσματικά αντιστοιχίες μεταξύ facts και rules. [23]

Στη προκειμένη περίπτωση χρήσης ενός conversational interface για σύμβουλο δραστηριοτήτων το NRules χρησιμοποιήθηκε για να βοηθήσει στην εξυπνότερη επιλογή tips που θα παρουσιάζονται στο χρήστη.

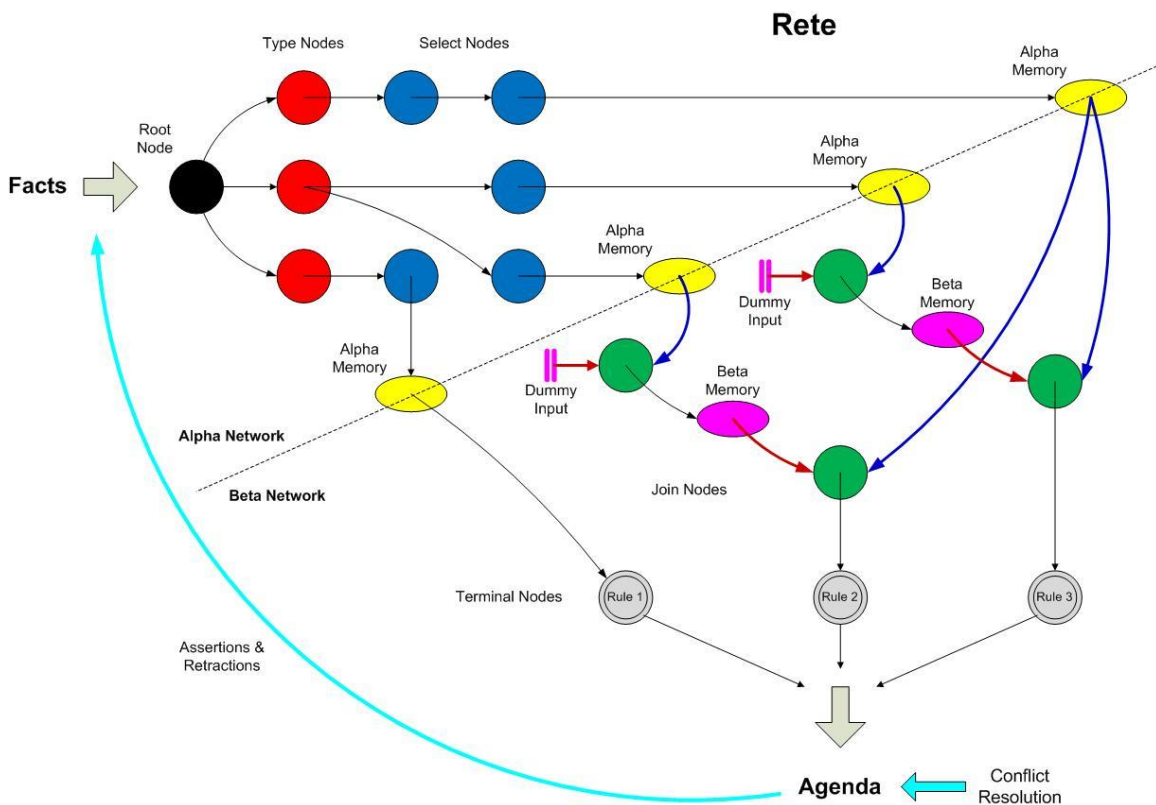
Το κάθε tip διαθέτει γνωρίσματα τα οποία το καθιστούν κατάλληλο μόνο για κάποιες συγκεκριμένες ομάδες χρηστών. Για παράδειγμα ένα tip μπορεί να διατίθεται μόνο σε χρήστες οι οποίοι είναι καπνιστές. Άλλα γνωρίσματα είναι η φυσική άσκηση, η ενυδάτωση του χρήστη, ο χρόνος ύπνου καθώς και το cluster στο οποίο βρίσκεστε με βάση τα προσωπικά του χαρακτηριστικά (Openness, Conscientiousness, Extroversion, Agreeableness και Neuroticism). Ο διαχωρισμός των tips που είναι εφαρμόσιμα σε κάποιο συγκεκριμένο χρήστη, με βάση τα χαρακτηριστικά και τις ασχολίες του, γίνεται από το NRules.

Rete

Ο αλγόριθμος Rete εφευρέθηκε από τον Dr. Charles L. Forgy του Carnegie Mellon University το 1979 και θεωρείται ευρέως ως ο αλγόριθμος που έκανε τα rule-based inferences αρκετά αποτελεσματικά ώστε να είναι πρακτικά στη χρήση. Είναι ένας pattern matching αλγόριθμος φτιαγμένος για rule based συστήματα.

Μία απλή εφαρμογή ενός αλγορίθμου συγκρίνει κάθε rule με κάθε fact στο knowledge base, εκτελώντας το rule όπου χρειάζεται, και ξεκινά ξανά μέχρι να τα εξαντλήσει. Αυτό τον καθιστά αργό στην εκτέλεση. Εδώ είναι που ο αλγόριθμος Rete είναι ιδιαίτερα χρήσιμος.

Ο αλγόριθμος Rete χτίζει ένα δίκτυο με κόμβους (nodes) όπου κάθε κόμβος (εκτός από το root) αντιστοιχεί σε ένα μοτίβο που εμφανίζεται στην αριστερή πλευρά ενός κανόνα αριστερού χεριού. Η διαδρομή από το root node προς ένα leaf node καθορίζει έναν πλήρη κανόνα αριστερά. Κάθε κόμβος έχει μνήμη γεγονότων (facts) που ικανοποιούν αυτό το μοτίβο. Καθώς επιβεβαιώνονται ή τροποποιούνται νέα facts, διαδίδονται κατά μήκος του δικτύου, προκαλώντας τους κόμβους να γίνουν annotated όταν αυτό το γεγονός ταιριάζει με αυτό το μοτίβο. Όταν ένα fact, ή συνδυασμός από facts προκαλεί την ικανοποίηση όλων των προτύπων για έναν δεδομένο κανόνα, φτάνουμε σε ένα leaf node και ενεργοποιείται ο αντίστοιχος κανόνας. Έτσι ο αλγόριθμος Rete είναι σχεδιασμένος για να θυσιάζει τη μνήμη για αυξημένη ταχύτητα και επιδόσεις. [24]



ILLUSTRATES THE ALPHA AND BETA NETWORKS AND BASIC NODE TYPES CONTAINED IN A RETE ΑΠΟ CHARLES YOUNG

Σχήμα 38: Αλγόριθμος Rete

Στο σχήμα απεικονίζεται η ροή του αλγόριθμου Rete. Τα facts μεταφέρονται μέσω των nodes, ξεκινώντας από το root node, στα διάφορα επίπεδα μνήμης όπου μετά συγκρίνονται με τα rules για να βρεθούν τα conflicts. Αυτό επαναλαμβάνετε μέχρι να καλυφθούν όλα τα διαθέσιμα στοιχεία.

4.1.6. Azure Functions

Το Azure Functions είναι ένα serverless solution που μας επιτρέπει να εφαρμόσουμε τη λογική του συστήματός μας σε άμεσα διαθέσιμα τμήματα κώδικα. Αυτά τα τμήματα κώδικα ονομάζονται functions (συναρτήσεις). Διαφορετικές λειτουργίες μπορούν να εκτελεστούν όποτε θέλουμε να ανταποκριθούμε σε κρίσιμα γεγονότα. Καθώς αυξάνονται τα requests, τα Azure Functions ικανοποιούν τη ζήτηση με όσους πόρους και λειτουργίες είναι απαραίτητες - αλλά μόνο όταν χρειάζεται. Όταν τα requests μειώνονται, τυχόν επιπλέον πόροι και εφαρμογές σταματούν αυτόματα. Αυτό το μοντέλο δίνει τη δυνατότητα στο χρήστη να χρεώνεται μόνο για τα resources τα οποία χρησιμοποιούν τα functions του, μόνο για το χρονικό διάστημα που τα functions εκτελούνται, και όχι όσο είναι απλά idle. [25]

Η χρήση των Azure Functions στη δεδομένη διπλωματική εργασία είναι για να στέλνουν tips στο χρήστη καθημερινά, χωρίς ο χρήστης να χρειάζεται να ξεκινά κάθε μέρα διάλογο με το bot. Τα Azure Functions κάνουν καθημερινά ένα request στο proactive messages endpoint το οποίο είναι υπεύθυνο να διαμοιράσει τα proactive messages σε όλους τους χρήστες.

4.1.7. Proactive Messages

Τα proactive messages παρέχουν τη δυνατότητα να αποστέλλονται μηνύματα από το bot χωρίς ο χρήστης να βρίσκεται σε μία συζήτηση. Αυτά τα μηνύματα έχουν συνήθως το χαρακτήρα notifications και στη προκειμένη περίπτωση χρησιμοποιούνται για να δίνουν στο χρήστη tips μια φορά τη μέρα. Για να ενεργοποιηθεί η αποστολή ενός τέτοιου μηνύματος, χρειάζεται ένα request σε ένα συγκεκριμένο endpoint, το οποίο πραγματοποιείτε αυτοματοποιημένα με τη χρήση Azure Functions.

4.1.8. Adaptive Cards

Τα Adaptive Cards είναι εργαλεία UI (User Interface) τα οποία δίνουν τη δυνατότητα στο προγραμματιστή να τα παραμετροποιήσει όπως αυτός επιθυμεί. Παρέχουν παραμετροποιήσιμο κείμενο καθώς και εικόνες ούτως ώστε η πληροφορία που περιέχουν να προβάλλεται στο χρήστη με ευχάριστο τρόπο. Υποστηρίζονται από το bot framework και έχουν παρόμοια εμφάνιση σε όλα τα κανάλια τα οποία υποστηρίζονται για επίτευξη συνοχής.

Περιγράφουν το περιεχόμενό τους ως απλό αντικείμενο JSON. Αυτό το περιεχόμενο μπορεί στη συνέχεια να αποδίδεται natively στο host application, όπου προσαρμόζεται αυτόματα στην εμφάνιση και την αίσθηση του host.

Αυτά που προσπαθούν να επιτύχουν είναι:

- Φορητές: Συνδέονται ευκολά με διάφορα UI frameworks
- Ανοιχτού κώδικα: Είναι open source με δυνατότητα να επέμβουμε στον κώδικα

- Χαμηλού κόστους: Εύκολες στον ορισμό και στο consumption τους από διάφορες εφαρμογές
- Εκφραστικές: Προσπαθούν να αποδώσουν με το καλύτερο τρόπο αυτό που θέλει ο προγραμματιστής
- Αμιγώς δηλωτικές: Δεν απαιτείται κώδικας ούτε επιτρέπεται
- Αυτόματο στίλ: Καθορίζεται αυτόματα από το UX (User Experience) της host εφαρμογής [26]

Η χρήση τους στο Activity Coaching Bot είναι σε δύο σημεία. Χρησιμοποιούνται για την παρουσίαση των αποτελεσμάτων των personal traits του χρήστη, τα οποία εμφανίζονται κάθε φορά που ολοκληρώνει ένα ερωτηματολόγιο και στοχεύουν να παρουσιάσουν σημαντικά στοιχεία του χρήστη με ευχάριστο τρόπο. Ο δεύτερος τρόπος χρήσης τους είναι για το ανέβασμα (uploading) των tips και ερωτηματολογίων. Ο χρήστης καλείτε να συμπληρώσει μια κάρτα για κάθε ερώτημα ή tip που θέλει να ανεβάσει στη βάση δεδομένων, καθώς και ζητείτε να δώσει το κωδικό διαχειριστή για να ανεβάσει δεδομένα στη βάση. Σε αυτή τη περίπτωση οι κάρτες στοχεύουν να προσφέρουν στο χρήστη ένα καλύτερο περιβάλλον UI για τη συμπλήρωση όλων των στοιχείων που χρειάζονται χωρίς να απαιτείτε εκτενής διάλογος με το bot για κάθε tip ή ερώτηση.

4.2. Εργαλεία / Λογισμικό

Για την δημιουργία του bot χρησιμοποιήθηκαν τα εξής εργαλεία:

- Microsoft Visual Studio
- Bot Application Template
- Bot Framework Emulator (V4)
- SourceTree
- Microsoft Azure

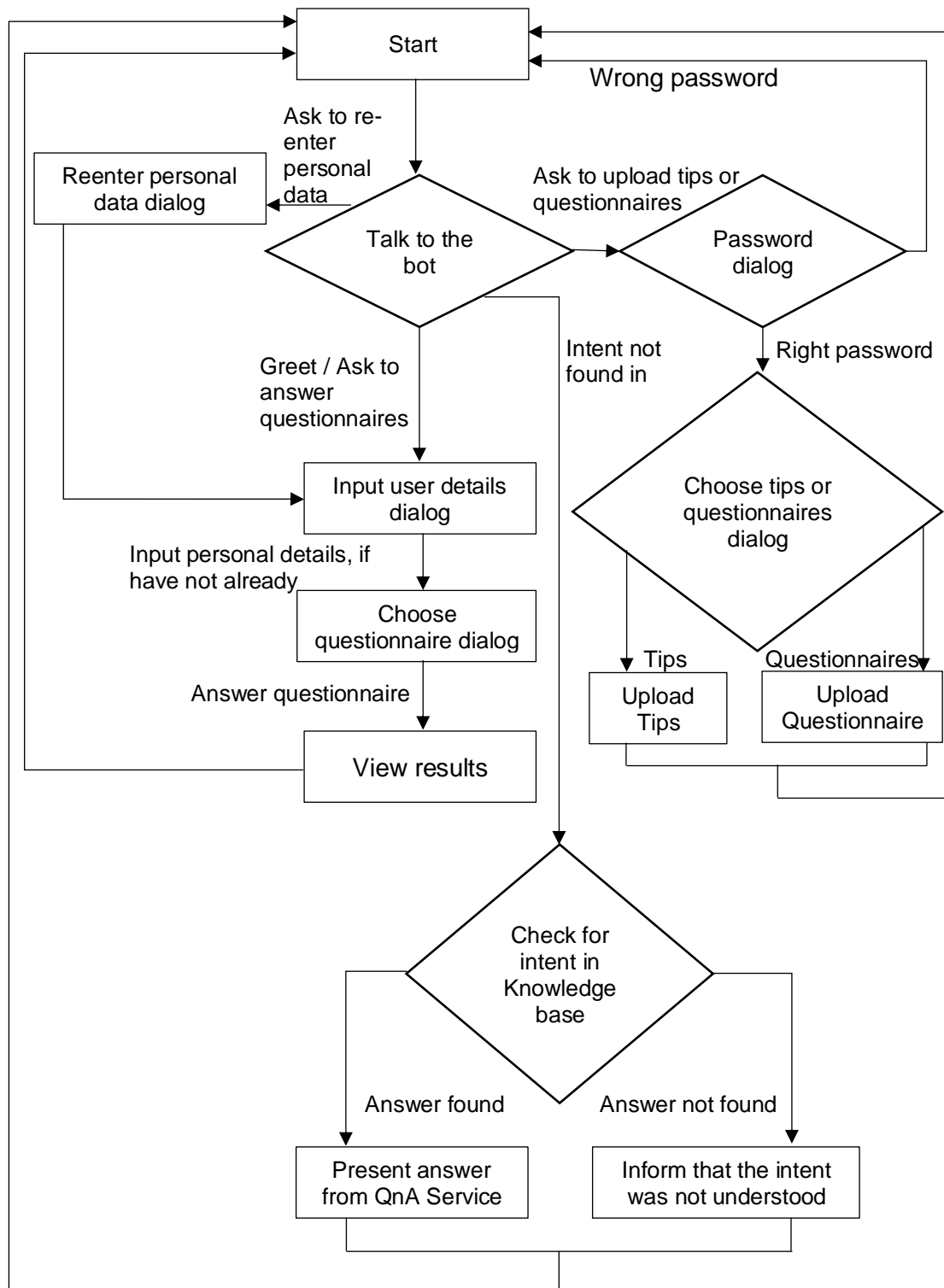
4.3. Υλοποίηση

Σε αυτό το σημείο αναλύεται η υλοποίηση του chatbot και ο τρόπος που όλες οι τεχνολογίες συνδέονται μεταξύ τους για να μπορεί να λειτουργήσει σωστά το bot. Πως και πότε επικοινωνεί με τη βάση δεδομένων, πως χειρίζεται το clustering, πότε στέλνει request στο LUIS και πότε στο QnA service, πότε προβάλλει Adaptive Cards και πώς χρησιμοποιεί το rules engine.

Επίσης αναλύεται ο τρόπος με τον οποίο στέλνονται τα tips μέσω proactive messages και πως παρέχεται η δυνατότητα να στέλνονται αυτόματα, καθημερινά.

Ο τρόπος υλοποίησης των διαλόγων από το bot framework κάνει χρήση του μοντέλου καταρράκτη (WaterfallDialog), το οποίο χωρίζει την διαδοχική ομιλία του χρήστη και του bot σε βήματα (WaterfallStep) τα οποία εκτελούνται πάντα σειριακά για την ολοκλήρωση του διαλόγου.

Το παρακάτω διάγραμμα περιγράφει τη ροή διαλόγου που χρησιμοποιεί το bot για να αντλήσει τα στοιχεία που χρειάζεται από το χρήστη. Τα ημερήσια tips παραστούνε στο χρήστη ανεξάρτητα με τη παρούσα θέση του στο διάγραμμα τη δεδομένη στιγμή, δεδομένου ότι έχει ήδη συμπληρωθεί τουλάχιστο ένα ερωτηματολόγιο.



Σχήμα 39: Flowchart Λειτουργίας Bot

Σε οποιοδήποτε βήμα του παραπάνω διαγράμματος, ο χρήστης έχει τη δυνατότητα να ζητήσει βοήθεια, και θα του εμφανιστεί το μήνυμα βοήθειας, ή να ζητήσει να εξέλθει από το δεδομένο διάλογο και να επιστρέψει κατευθείαν στην αρχή του διαγράμματος.

4.3.1. MainDialog.cs

Αυτός ο διάλογος είναι ο βασικός διάλογος του bot. Με το που εκκινήσει η λειτουργία του bot, ο χρήστης μπαίνει σε αυτό το διάλογο. Όταν ο χρήστης ολοκληρώσει μία διαδικασία, για παράδειγμα απαντήσει ένα ερωτηματολόγιο, επιστρέφει σε αυτό το διάλογο. Επίσης όποτε ένας διάλογος διακοπεί από το χρήστη, το bot επιστρέφει και πάλι σε αυτό το διάλογο.

Η λειτουργία αυτού του διαλόγου είναι να παρουσιάζει στο χρήστη το αρχικό μήνυμα και μετά να τον παραπέμπει στο διάλογο τον οποίο χρειάζεται, ή να του επιστρέφει το κατάλληλο μήνυμα. Αυτό γίνεται μέσα στο `ActStepAsync Task`. Στην αρχή αυτού του `Task` καλούνται τα δεδομένα του χρήστη από τη βάση δεδομένων και στη συνέχεια αποστέλλετε το `utterance` του χρήστη στο LUIS. Αφού επιστραφεί το `intent` του χρήστη, υπάρχει ένα `switch statement` το οποίο παραπέμπει το χρήστη στον αντίστοιχο διάλογο. Αν το `intent` του χρήστη δε καλύπτετε από αυτά που υπάρχουν στο LUIS τότε σαρώνετε το `utterance` για το keyword “help” και αν υπάρχει, σημαίνει ότι ο χρήστης χρειάζεται βοήθεια και του επιστρέφεται το μήνυμα βοήθειας. Αν όχι, γίνεται `request` στο `knowledge base` το οποίο περιέχει το `chit-chat module` με συχνές ερωτήσεις και απαντήσεις. Αν το `utterance` του χρήστη αντιστοιχεί σε κάποια ερώτηση, τότε επιστρέφεται η απάντησή της. Αν και πάλι δε βρεθεί κανένας πιθανός τρόπος να απαντηθεί το `utterance` του χρήστη, τότε επιστρέφεται μήνυμα το οποίο ενημερώνει ότι το αίτημα δεν ήταν κατανοητό και ο διάλογος ξεκινάει ξανά από την αρχή. Σε οποιαδήποτε περίπτωση το bot δε χρειάζεται να παραπέμπει το χρήστη σε άλλο διάλογο και απλά του απαντά με ένα μήνυμα, στη συνέχεια και πάλι ξεκινάει το βασικό διάλογο (`MainDialog.cs`) από την αρχή για να συνεχίσει η συζήτηση με το χρήστη.

```

// Call LUIS and prompt the corresponding dialog. Search the knowledge base if intent is not found.
var luisResult = await _luisRecognizer.RecognizeAsync<LuisModel>(stepContext.Context, cancellationToken);
switch (luisResult.TopIntent().intent)
{
    case LuisModel.Intent.AddQuestionnairesOrTips:
        return await stepContext.BeginDialogAsync(nameof(AddQuestionnairesOrTipsDialog), PersonalDetailsDialog.PersonalDetails, cancellationToken);

    case LuisModel.Intent.AnswerQuestionnaires:
        return await stepContext.BeginDialogAsync(nameof(AnswerQuestionnairesDialog), PersonalDetailsDialog.PersonalDetails, cancellationToken);

    case LuisModel.Intent.Greet:
        if (PersonalDetailsDialog.PersonalDetails.Name == null)
            PersonalDetailsDialog.PersonalDetails.Name = luisResult.Entities.personName != null ? char.ToUpper(luisResult.Entities.personName[0][0]) +
                luisResult.Entities.personName[0].Substring(1) : null;
        if (PersonalDetailsDialog.PersonalDetails.Age == null)
            PersonalDetailsDialog.PersonalDetails.Age = (int?)luisResult.Entities.age?[0].Number;

        // Greeting message
        var greetText = (PersonalDetailsDialog.PersonalDetails.Name == null ? Response.Greet() : Response.Greet(PersonalDetailsDialog.PersonalDetails.Name));
        var greetTextMessage = MessageFactory.Text(greetText, greetText, InputHints.IgnoringInput);
        await stepContext.Context.SendActivityAsync(greetTextMessage, cancellationToken);

        // Run the PersonalDetailsDialog giving it whatever details we have from the LUIS call, it will fill out the remainder.
        return await stepContext.BeginDialogAsync(nameof(PersonalDetailsDialog), PersonalDetailsDialog.PersonalDetails, cancellationToken);

    case LuisModel.Intent.EnterPersonalDetails:
        if (CheckDetails())
            return await stepContext.BeginDialogAsync(nameof(ReenterDetailsDialog), PersonalDetailsDialog.PersonalDetails, cancellationToken);
        else
            return await stepContext.BeginDialogAsync(nameof(PersonalDetailsDialog), PersonalDetailsDialog.PersonalDetails, cancellationToken);

    default:
        // Catch all for unhandled intents
        // Try to find an answer on the knowledge base
        var knowledgeBaseResult = await _luisRecognizer.SampleQnA.GetAnswersAsync(stepContext.Context);

        if (Regex.IsMatch(stepContext.Context.Activity.Text, "help", RegexOptions.IgnoreCase))
        {
            // If intent is help, return help message
            var helpMessageText = Response.HelpMessage();
            return await stepContext.ReplaceDialogAsync(InitialDialogId, helpMessageText, cancellationToken);
        }
        else if (knowledgeBaseResult?.FirstOrDefault() != null)
            return await stepContext.ReplaceDialogAsync(InitialDialogId, knowledgeBaseResult[0].Answer, cancellationToken);
        else
        {
            // If it's not on the knowledge base, return error message
            var didntUnderstandMessageText = $"Sorry, I didn't get that. Please try asking in a different way (intent was {luisResult.TopIntent().intent})";
            var didntUnderstandMessage = MessageFactory.Text(didntUnderstandMessageText, didntUnderstandMessageText, InputHints.IgnoringInput);
            await stepContext.Context.SendActivityAsync(didntUnderstandMessage, cancellationToken);
        }
        break;
}
return await stepContext.NextAsync(null, cancellationToken);

```

Σχήμα 40: LUIS Intent Switch

Σε περίπτωση που επιστρέψουμε στο βασικό διάλογο από άλλο διάλογο, εμφανίζεται το μήνυμα "What else can I do for you?" μέσω του FinalStepAsync Task πριν επανεκκινηθεί ο διάλογος.

Ο βασικός διάλογος (MainDialog.cs) εμπεριέχει τους ορισμούς όλων των υπολοίπων διαλόγων που χρησιμοποιούνται από το bot. Επίσης εδώ βρίσκονται σημαντικά functions τα οποία κάνουν διάφορες λειτουργίες που χρειάζονται να καλούνται είτε από αυτό το διάλογο, είτε από άλλους. Εδώ διεξάγεται το μεγαλύτερο ποσοστό της επικοινωνίας με τη βάση δεδομένων.

Το WriteToDB function είναι υπεύθυνο να στέλνει τα δεδομένα του χρήστη στη βάση δεδομένων. Στο ασύγχρονο function WriteAsync δε χρησιμοποιείται η await, για να μπορεί ο χρήστης να συνεχίσει τη συζήτησή του με το bot όσο τα δεδομένα του αποστέλλονται στη βάση δεδομένων, ούτως ώστε το bot να είναι πιο responsive. Αυτό είναι ιδιαίτερα σημαντικό διότι αυτό το function καλείται κάθε φορά που ο χρήστης απαντά μια ερώτηση σε ένα ερωτηματολόγιο, για να παρέχεται η δυνατότητα στο χρήστη να μπορεί να διακόψει το ερωτηματολόγιο χωρίς να χρειάζεται να το ξεκινήσει από την αρχή στο μέλλον. Αυτό σημαίνει ότι η χρήση της await θα έκανε το bot εσθίτα πιο αργό.

```
// Sends Personal Data to database
10 references | demetrisbakas, 25 days ago | 1 author, 2 changes | 0 exceptions
public static async void WriteToDB(WaterfallStepContext stepContext, CancellationToken cancellationToken)
{
    // Send to DB
    var changes = new Dictionary<string, object>() { { PersonalDetailsDialog.PersonalDetails.UserID, PersonalDetailsDialog.PersonalDetails } };
    try
    {
        #pragma warning disable CS4014 // Because this call is not awaited, execution of the current method continues before the call is completed
        CosmosDBQuery.WriteAsync(changes, cancellationToken);
        #pragma warning restore CS4014 // Because this call is not awaited, execution of the current method continues before the call is completed
    }
    catch (Exception e)
    {
        await stepContext.Context.SendActivityAsync($"Error while connecting to database.\n\n{e}");
    }
}
```

Σχήμα 41: Αποστολή Προσωπικών Δεδομένων στη Βάση Δεδομένων

Με τον ίδιο τρόπο λειτουργούν η SendTipsToDB η οποία στέλνει τα καινούρια tips στη βάση δεδομένων, καθώς και η SendQuestionnairesToDB, η οποία στέλνει τα καινούρια ερωτηματολόγια στη βάση δεδομένων.

Η QueryTipsAsync είναι υπεύθυνη να λαμβάνει όλα τα tips από τη βάση δεδομένων για να δοθούν στο rules engine το οποίο θα αποφασίσει πια tips μπορούν να παρουσιαστούν στο χρήστη.

Η QueryQuestionnairesAsync, παρομοίως με τη QueryTipsAsync λαμβάνει όλα τα ερωτηματολόγια από τη βάση δεδομένων για να παρουσιαστούν στο χρήστη.

Η QueryClusterDetailsAsync είναι υπεύθυνη να μαζεύει τα personality traits όλων των χρηστών για να πραγματοποιηθεί το clustering. Στο τέλος καλεί την RemoveNullValues, η οποία αφαιρεί όσα στοιχεία χρηστών είναι null. Αν κάποιος χρήστης δεν έχει ολοκληρώσει την εισαγωγή των στοιχείων του, κάποια από αυτά θα είναι null.

```
// Requests clustering data from the database
1 reference | demetrisbakas, 25 days ago | 1 author, 4 changes | 0 exceptions
public static async Task<List<ClusterPersonalDetailsWithoutNull>> QueryClusterDetailsAsync()
{
    var sqlQueryText = "SELECT c.document.Extraversion, c.document.Agreeableness, c.document.Conscientiousness, c.document.Neuroticism, c.document.Openness FROM c";
    QueryDefinition queryDefinition = new QueryDefinition(sqlQueryText);
    CosmosClient cosmosClient = new CosmosClient(cosmosServiceEndpoint, cosmosDBKey);
    Azure.Cosmos.Database database;
    database = await cosmosClient.CreateDatabaseIfNotExistsAsync(cosmosDBDatabaseName);
    Container container;
    container = await database.CreateContainerIfNotExistsAsync(cosmosDBContainerId, "/id");
    FeedIterator<ClusterPersonalDetails> queryResultSetIterator = container.GetItemQueryIterator<ClusterPersonalDetails>(queryDefinition);
    List<ClusterPersonalDetails> detailsList = new List<ClusterPersonalDetails>();
    while (queryResultSetIterator.HasMoreResults)
    {
        Azure.Cosmos.FeedResponse<ClusterPersonalDetails> currentResultSet = await queryResultSetIterator.ReadNextAsync();
        foreach (ClusterPersonalDetails details in currentResultSet)
        {
            detailsList.Add(details);
        }
    }
    return RemoveNullValues(detailsList);
}
```

Σχήμα 42: Συλλογή Personality Traits για Clustering από τη Βάση Δεδομένων

Επίσης υπάρχει το function, με όνομα ClusteringAsync, το οποίο διεξάγει το clustering μέσω ML.NET, με τη χρήση του αλγορίθμου K-Means.

```

// Clusters all the users based on their personality traits
1 reference | demetrisbakas, 25 days ago | 1 author, 7 changes | 0 exceptions
public static async Task<int> ClusteringAsync()
{
    var dataList = await ClusteringData;

    var context = new MLContext();

    IDataView data = context.Data.LoadFromEnumerable(dataList);

    var trainTestData = context.Data.TrainTestSplit(data, testFraction: 0.2);

    var pipeline = context.Transforms.Concatenate("Features", "Extraversion", "Agreeableness", "Conscientiousness", "Neuroticism", "Openness").Append
        (context.Clustering.Trainers.KMeans(featureColumnName: "Features", numberOfClusters: 5));

    var model = pipeline.Fit(trainTestData.TrainSet);

    var predictionFunc = context.Model.CreatePredictionEngine<ClusterPersonalDetailsWithoutNull, SeedPrediction>(model);

    var prediction = predictionFunc.Predict(new ClusterPersonalDetailsWithoutNull
    {
        Extraversion = PersonalDetailsDialog.PersonalDetails.Extraversion != null ? (float)PersonalDetailsDialog.PersonalDetails.Extraversion : 0,
        Agreeableness = PersonalDetailsDialog.PersonalDetails.Agreeableness != null ? (float)PersonalDetailsDialog.PersonalDetails.Agreeableness : 0,
        Conscientiousness = PersonalDetailsDialog.PersonalDetails.Conscientiousness != null ? (float)PersonalDetailsDialog.PersonalDetails.Conscientiousness : 0,
        Neuroticism = PersonalDetailsDialog.PersonalDetails.Neuroticism != null ? (float)PersonalDetailsDialog.PersonalDetails.Neuroticism : 0,
        Openness = PersonalDetailsDialog.PersonalDetails.Openness != null ? (float)PersonalDetailsDialog.PersonalDetails.Openness : 0
    });

    return prediction.SelectedClusterId;
}

```

Σχήμα 43: Υλοποίηση Clustering

Το function `ReplaceMissingQuestionValues` είναι υπεύθυνο να συμπληρώνει τις απαντήσεις σε ερωτήσεις που δεν έχουν δικές τους απαντήσεις, από τα ερωτηματολόγια. Συμπληρώνονται με τις default απαντήσεις του `QuestionTopFive` class.

Τέλος το `CheckDetails` function ελέγχει αν ο χρήστης έχει ήδη εισάγει όλα τα προσωπικά του δεδομένα.

4.3.2. PersonalDetailsDialog.cs

Αυτός ο διάλογος είναι υπεύθυνος να συλλέγει όλα τα προσωπικά δεδομένα του χρήστη. Μόλις η συλλογή των στοιχείων ολοκληρωθεί με επιτυχία, ξεκινάει τον διάλογο `QuestionnaireChoiceDialog.cs` στον οποίο ο χρήστης καλείται να επιλέξει ερωτηματολόγιο.

Τα στοιχεία του χρήστη αποθηκεύονται στη μεταβλητή `PersonalDetails`, η οποία είναι τύπου `PersonalDetails`. Το συγκεκριμένο class αναλύεται στη συνέχεια.

Μετά την ολοκλήρωση κάθε βήματος τα στοιχεία του χρήστη αποστέλλονται στη βάση δεδομένων ούτως ώστε να μη χρειαστεί να ξανά-συμπληρωθούν από το χρήστη αν χρειαστεί να φύγει από το συγκεκριμένο διάλογο πριν αυτός ολοκληρωθεί.

Αρχικά, ο διάλογος ελέγχει αν το όνομα του χρήστη έχει ήδη δοθεί. Αυτό μπορεί να συμβαίνει λόγω του ότι ο χρήστης έχει ήδη δώσει το όνομά του αλλά βγήκε από το διάλογο και τώρα θέλει να συμπληρώσει τα υπόλοιπα στοιχεία του. Ένας άλλος τρόπος είναι να δώσει το όνομά του όταν χαιρετάει το bot για πρώτη φορά, οπότε το bot δε χρειάζεται να ρωτήσει το χρήστη για το όνομα. Ο πιο συχνός τρόπος πρόσβασης του ονόματος είναι μέσω του καναλιού. Για παράδειγμα, αν ο χρήστης χρησιμοποιεί το Microsoft Teams ως κανάλι επικοινωνίας του με το bot, τότε το bot θα αντλήσει το ονοματεπώνυμο του χρήστη από το όνομα που έχει στο λογαριασμό του στο Microsoft Teams.

Στη συνέχεια ο χρήστης τίθεται να δώσει την ηλικία του, αν δε την έχει δώσει ήδη. Παρέχεται και η δυνατότητα να δώσει την ηλικία του όταν χαιρετά το bot για πρώτη φορά. Αν δεν έχει δώσει με κάποιο άλλο τρόπο την ηλικία του, τη δίνει μόλις ερωτηθεί, είτε σε μορφή αριθμού, είτε λεκτικά σαν πρόταση.

Μετάπειτα ο χρήστης ερωτάτε για το φύλο του και με τη χρήση του ChoicePrompt έχει να επιλέξει μεταξύ δύο επιλογών, “Male” και “Female”. Υπάρχει η δυνατότητα να γράψει την απάντησή του σε μορφή κειμένου, αντί να επιλέξει μία από τις δύο επιλογές. Αν αυτό που έγραψε δεν είναι το ίδιο με κάποια από τις πιθανές απαντήσεις, τότε το bot επαναλαμβάνει την ερώτηση.

Στη συνέχεια ερωτάτε αν είναι καπνιστής και δίνονται οι απαντήσεις Yes και No. Η ερώτηση γίνεται με τη χρήση ConfirmPrompt και παρέχονται όλες οι δυνατότητες που παρέχονται και στη προηγούμενη ερώτηση. Αν ο χρήστης απαντήσει κάτι που δεν είναι αποδεκτό, η ερώτηση θα επαναληφθεί.

Έπειτα ο χρήστης ερωτάτε για τη ποσότητα νερού που καταναλώνει καθημερινά σε ποτήρια, τις ώρες που κοιμάται το βράδυ και τις ώρες φυσικής άσκησης που ολοκληρώνει κάθε εβδομάδα. Μπορεί να απαντήσει είτε με αριθμό, είτε λεκτικά.

Τέλος υπάρχει το ConfirmStepAsync στο οποίο παρουσιάζονται όλα τα δεδομένα που συλλέχθηκαν από το bot στο χρήστη και ερωτάτε αν είναι σωστά με τη χρήση ενός ConfirmPrompt. Αν ο χρήστης απαντήσει αρνητικά τότε επανεκκινείται ο διάλογος και ο χρήστης εισάγει ξανά τα στοιχεία του. Αν απαντήσει θετικά τότε μεταφέρεται στο διάλογο QuestionnaireChoiceDialog.cs για να επιλέξει ερωτηματολόγιο.

Τα ConfirmPrompt και ChoicePrompt που χρησιμοποιούνται διαθέτουν validators σε περίπτωση που κάποιο από τα δεδομένα δεν συμβαδίζει με αυτά που είναι επιτρεπτά από το prompt. Παρόλα αυτά, για το TextPrompt κάτι τέτοιο δε διατίθεται. Οπότε δημιουργήθηκε ένας custom validator ο οποίος καλύπτει τις ανάγκες αυτού του prompt. Με τη χρήση ενός switch statement και ενός enum που ο validator παίρνει σαν όρισμα, αντιλαμβάνεται ποιο είδος validation καλείτε να χρησιμοποιήσει. Στόχος του είναι να κάνει τη συλλογή ονόματος και ηλικίας εξυπνότερες, καθώς στέλνει αυτά που γράφει ο χρήστης στο LUIS και ελέγχει αν ο χρήστης έχει απαντήσει επιτυχώς την ερώτηση. Μετέπειτα, στο πεδίο της ηλικίας, ψάχνει με χρήση regular expression για αριθμό μέσα σε αυτά που έγραψε ο χρήστης για να τον θέσει ως ηλικία.

```
1 reference | demetrisbakas, 118 days ago | 1 author, 1 change | 0 exceptions
private async Task<bool> TextPromptValidatorAsync(PromptValidatorContext<string> promptContext, CancellationToken cancellationToken)
{
    switch (promptContext.Options.Validations != null ? (Validator)promptContext.Options.Validations : (Validator)(-1))
    {
        case Validator.Name:
            luisResult = await MainDialog.Get_luisRecognizer().RecognizeAsync<LuisModel>(promptContext.Context, cancellationToken);
            PersonalDetails.Name = (luisResult.Entities.personName != null ? char.ToUpper(luisResult.Entities.personName[0][0]) + luisResult.Entities.personName[0].Substring(1) :
            null);

            if (PersonalDetails.Name == null)
                return await Task.FromResult(false);
            else
                return await Task.FromResult(true);

        case Validator.Age:

            luisResult = await MainDialog.Get_luisRecognizer().RecognizeAsync<LuisModel>(promptContext.Context, cancellationToken);
            if (luisResult.Entities.age != null)
                PersonalDetails.Age = (int?)luisResult.Entities.age[0].Number;
            else if (Regex.Match(promptContext.Context.Activity.Text, @"^\d+$").Value != "")
                // Second check, just in case. Returns null if parse fails
                PersonalDetails.Age = Int32.TryParse(Regex.Match(promptContext.Context.Activity.Text, @"^\d+$").Value, out var tempVal) ? tempVal : (int?)null;

            if (PersonalDetails.Age == null)
                return await Task.FromResult(false);
            else
                return await Task.FromResult(true);

        default:
            return await Task.FromResult(true);
    }
}
```

Σχήμα 44: Text Prompt Validator

Η συνάρτηση ClearDetails εξισώνει με null όλα τα προσωπικά δεδομένα του χρήστη σε περίπτωση που αυτός επιθυμεί να τα εισάγει ξανά.

4.3.3. QuestionnaireChoiceDialog.cs

Αυτός ο διάλογος δίνει τη δυνατότητα στο χρήστη να επιλέξει πιο ερωτηματολόγιο επιθυμεί να απαντήσει.

Στο ChooseQuestionnaireStepAsync Task αντλούνται όλα τα ερωτηματολόγια από τη βάση δεδομένων. Οι τίτλοι των ερωτηματολογίων δίνονται σαν απαντήσεις στο ChoicePrompt στο οποίο ο χρήστης καλείτε να επιλέξει το ερωτηματολόγιό του.

Όταν η επιλογή ερωτηματολογίου γίνει επιτυχώς, το ερωτηματολόγιο που έχει επιλεγεί φορτώνεται και στέλνεται στο διάλογο TopFiveDialog.cs ο οποίος είναι υπεύθυνος να παρουσιάσει το ερωτηματολόγιο στο χρήστη.

```
1 reference | demetrisbakas, 272 days ago | 1 author, 1 change | 0 exceptions
private async Task<DialogTurnResult> ChooseQuestionnaireStepAsync(WaterfallStepContext stepContext, CancellationToken cancellationToken)
{
    var choiceMsgText = MainDialog.Response.ChooseQuestionnaire();
    var promptMessage = MessageFactory.Text(choiceMsgText, choiceMsgText, InputHints.ExpectingInput);
    var retryText = $"Please choose one option.\n\n{choiceMsgText}";
    var retryPromptText = MessageFactory.Text(retryText, retryText, InputHints.ExpectingInput);

    var questionnaireChoice = new List<Choice>();
    var questionnairesNames = new List<string>();

    // Wait for the questionnaires to be fetched from the database
    MainDialog.Response.Questionnaires = await MainDialog.Questionnaires;

    foreach (KeyValuePair<string, List<QuestionTopFive>> kvp in MainDialog.Response.Questionnaires)
    {
        questionnairesNames.Add(kvp.Key);
    }
    foreach (string name in questionnairesNames)
    {
        questionnaireChoice.Add(new Choice(name));
    }

    return await stepContext.PromptAsync(nameof(ChoicePrompt), new PromptOptions { Prompt = promptMessage, Choices = questionnaireChoice, RetryPrompt = retryPromptText, Style = ListStyle.HeroCard }, cancellationToken);
}

1 reference | demetrisbakas, 272 days ago | 1 author, 2 changes | 0 exceptions
private async Task<DialogTurnResult> FinalStepAsync(WaterfallStepContext stepContext, CancellationToken cancellationToken)
{
    activeQuestionnaire = (from kvp in MainDialog.Response.Questionnaires where kvp.Key == ((FoundChoice)stepContext.Result).Value select kvp.Value).FirstOrDefault();

    return await stepContext.BeginDialogAsync(nameof(TopFiveDialog), PersonalDetailsDialog.PersonalDetails, cancellationToken);
}
```

Σχήμα 45: Επιλογή Ερωτηματολογίου

4.3.4. TopFiveDialog.cs

Ο τρόπος λειτουργίας αυτού του διαλόγου είναι μια προσομοίωση ενός loop. Στο AskQuestionStepAsync Task υπάρχει ένα foreach loop το οποίο διατρέχει όλες τις ερωτήσεις του ερωτηματολογίου. Για κάθε ερώτηση, ελέγχει στα προσωπικά δεδομένα του χρήστη αν η ερώτηση έχει ήδη απαντηθεί. Αν έχει απαντηθεί προχωρά στην επόμενη, αν όχι, τότε η ερώτηση εμφανίζεται στο χρήστη σε μορφή ChoicePrompt.

Στη συνέχεια, το FinalStepAsync Task ελέγχει αν το ερωτηματολόγιο έχει τελειώσει. Αν όχι, τότε στέλνει το αποτέλεσμα στη βάση δεδομένων και ξεκινά τον διάλογο TopFiveDialog από την αρχή για να εμφανιστεί η επόμενη ερώτηση.

```
else
{
    // Adding 1 to the answers index because it starts from 0
    PersonalDetailsDialog.PersonalDetails.QuestionnaireAnswers.Add(activeQuestion, ++((FoundChoice)stepContext.Result).Index);
    MainDialog.WriteToDB(stepContext, cancellationToken);

    return await stepContext.BeginDialogAsync(nameof(TopFiveDialog), PersonalDetailsDialog.PersonalDetails, cancellationToken);
}
```

Σχήμα 46: Επανάληψη του διαλόγου TopFiveDialog

Όταν το ερωτηματολόγιο τελειώσει, καλείτε το CalculatePersonalityTraitsAsync Task το οποίο είναι υπεύθυνο να υπολογίσει τα καινούρια personality traits του χρήστη, και στη συνέχεια κατασκευάζεται και προβάλλεται το adaptive card με τα αποτελέσματα

του χρήστη στα personality traits. Μόλις προβληθεί, ο διάλογος τελειώνει και ο χρήστης επιστρέφει πίσω στο MainDialog.

```
if (finished)
{
    await CalculatePersonalityTraitsAsync();
    string tip = await MainDialog.Response.TipMessageAsync();

    MainDialog.WriteToDB(stepContext, cancellationToken);

    var resultCard = CreateAdaptiveCardAttachment();
    var response = MessageFactory.Attachment(resultCard/*, ssml: "Here are your results!"*/);
    await stepContext.Context.SendActivityAsync(response, cancellationToken);

    // Resetting the flag, in case new user comes
    finished = false;
    finishedBefore = true;
    return await stepContext.EndDialogAsync(PersonalDetailsDialog.PersonalDetails, cancellationToken);
}
```

Σχήμα 47: Υπολογισμός Score Personality Traits

Αν το ερωτηματολόγιο έχει ήδη απαντηθεί από πριν, τότε δεν εμφανίζεται καμία ερώτηση και προβάλλεται ανάλογο μήνυμα. Έπειτα ο χρήστης επιστρέφει στο MainDialog.

```
// Send a message the this questionnaire has already been finished
if (finishedBefore)
{
    var finishedText = MainDialog.Response.FinishedQuestionnaire();
    var finishedTextMessage = MessageFactory.Text(finishedText, finishedText, InputHints.IgnoringInput);
    await stepContext.Context.SendActivityAsync(finishedTextMessage, cancellationToken);
}
```

Σχήμα 48: Έλεγχος αν το Ερωτηματολόγιο έχει ήδη Ολοκληρωθεί

Το CalculatePersonalityTraitsAsync Task υπολογίζει τα personality traits του χρήστη με βάση τις απαντήσεις του στα ερωτηματολόγια. Η κάθε ερώτηση που απαντάτε από το χρήστη, αποθηκεύεται στα δεδομένα του χρήστη μαζί με την απάντηση την οποία έδωσε, ούτως ώστε να μη χρειαστεί να επαναληφθεί αυτή η ερώτηση από το bot. Το CalculatePersonalityTraitsAsync μαζεύει όλες τις ερωτήσεις από όλα τα ερωτηματολόγια που υπάρχουν στη βάση δεδομένων. Κοιτάζει αν κάθε ερώτηση υπάρχει στα προσωπικά δεδομένα του χρήστη. Για κάθε ερώτηση που είναι απαντημένη, υπολογίζεται το score και το προσθέτει στο ανάλογο personality trait. Μόλις εξαντληθούν όλες οι ερωτήσεις, βρίσκει τους μέσους όρους (averages) του κάθε personality trait και τους αποθηκεύει στα προσωπικά στοιχεία του χρήστη. Ο λόγος που υπολογίζεται ο μέσος όρος κάθε personality trait, είναι διότι όταν προστεθούν καινούρια ερωτηματολόγια στη βάση δεδομένων, υπάρχει η περίπτωση να υπάρχουν περισσότερες ερωτήσεις για ένα personality trait σε σχέση με τα υπόλοιπα. Χωρίς το μέσω όρο, τα αποτελέσματα θα έδειχναν εκείνο το personality trait να έχει υψηλότερη βαθμολογία από τα υπόλοιπα, το οποίο δε θα ήταν απαραίτητα ορθό.

Στη συνέχεια καλείτε η συνάρτηση MainDialog.ClusteringAsync() η οποία διεξάγει το clustering και επιστρέφει το cluster του δεδομένου χρήστη, το οποίο αποθηκεύεται στα στοιχεία του.

4.3.5. ReenterDetailsDialog.cs


Αυτός ο διάλογος ρωτά το χρήστη αν θέλει να εισάγει ξανά τα προσωπικά του δεδομένα με τη χρήση ενός ConfirmPrompt. Αν ο χρήστης απαντήσει θετικά, καλείτε η συνάρτηση PersonalDetailsDialog.ClearDetails(), η οποία εξισώνει τα προσωπικά δεδομένα του χρήστη με null. Στη συνέχεια εκκινεί ο διάλογος PersonalDetailsDialog. Αν ο χρήστης απαντήσει αρνητικά, ο διάλογος ολοκληρώνεται και ο χρήστης επιστρέφει πίσω στο MainDialog.

4.3.6. AuthenticationDialog.cs

Ο σκοπός αυτού του διαλόγου είναι η εισαγωγή κωδικού για να γίνει το authentication του χρήστη. Το authentication αυτό γίνεται όταν ο χρήστης επιχειρεί να ανεβάσει ερωτηματολόγια ή tips στη βάση δεδομένων.

Για την εισαγωγή του κωδικού γίνεται χρήση ενός adaptive card το οποίο έχει παραμετροποιηθεί με το AdaptiveCardPrompt class για να λειτουργεί σαν διάλογος. Αυτό σημαίνει ότι, όταν ο χρήστης πατήσει το κουμπί “Submit” ο διάλογος θα συνεχιστεί, χωρίς να χρειάζεται ο χρήστης να πει κάτι άλλο στο bot.

Η περιγραφή του adaptive card γίνεται στο αρχείο passphrase.json και διαβάσετε με τη χρήση του PrepareCard class. Εμπεριέχει ένα TextBlock το οποίο ζητά από το χρήστη να εισάγει τον κωδικό, το Input.Text, που είναι το textbox στο οποίο ο χρήστης εισάγει τον κωδικό και το Action.Submit που είναι το κουμπί για να γίνει επικύρωση του κωδικού.



Σχήμα 49: Κάρτα Εισαγωγής Κωδικού Διαχειριστή στον Emulator

Ο κωδικός φυλάσσεται στη μεταβλητή με το όνομα “password” σε μορφή MD5. Η συνάρτηση MD5Hash μετατρέπει το αλφαριθμητικό που δίνει ο χρήστης σε MD5 για να μπορεί να γίνει η σύγκριση των δύο αλφαριθμητικών κωδικών.

Αν ο κωδικός που δόθηκε είναι ορθός, τότε ο χρήστης παραπέμπεται στο διάλογο UploadTipsOrQuestionnairesDialog όπου θα μπορεί να επιλέξει αν επιθυμεί να ανεβάσει tips ή ερωτηματολόγιο. Αν ο κωδικός είναι λανθασμένος, τότε εμφανίζεται κατάλληλο μήνυμα και ο διάλογος τερματίζεται επιστρέφοντας στο MainDialog.

```

var result = JsonConvert.DeserializeObject<PassJson>(stepContext.Result.ToString());
if (password == MD5Hash(result.Pass))
{
    return await stepContext.BeginDialogAsync(nameof(UploadTipsOrQuestionnairesDialog), PersonalDetailsDialog.PersonalDetails, cancellationToken);
}
else
{
    var messageText = MainDialog.Response.WrongPassword();
    var promptMessage = MessageFactory.Text(messageText, messageText, InputHints.ExpectingInput);
    await stepContext.PromptAsync(nameof(TextPrompt), new PromptOptions { Prompt = promptMessage }, cancellationToken);
    return await stepContext.EndDialogAsync(PersonalDetailsDialog.PersonalDetails, cancellationToken);
}

```

Σχήμα 50: Μετατροπή Κωδικού σε MD5 Hash

4.3.7. UploadTipsOrQuestionnairesDialog.cs

Αυτός ο διάλογος δίνει την ευκαιρία στο χρήστη να επιλέξει αν θέλει να ανεβάσει tips ή ερωτηματολόγιο στη βάση δεδομένων με τη χρήση ενός ChoicePrompt. Αν ο χρήστης απαντήσει tips, τότε παραπέμπεται στο διάλογο NumberOfTipsDialog, όπου τίθεται να δώσει τον αριθμό των tips που επιθυμεί να ανεβάσει, καθώς αν επιλέξει ερωτηματολόγιο παραπέμπεται στο NameOfQuestionnaireDialog όπου τίθεται να ονομάσει το ερωτηματολόγιό του.

4.3.8. NumberOfTipsDialog.cs

Σε αυτό το διάλογο ο χρήστης ερωτάτε για τον αριθμό των tips τα οποία επιθυμεί να ανεβάσει στη βάση δεδομένων. Μόλις ο χρήστης δώσει τον αριθμό, παραπέμπεται στο διάλογο UploadTipsDialog για να ξεκινήσει η εισαγωγή των tips.

Σε αυτό το class αποθηκεύεται ο αριθμός των tips, καθώς και μια λίστα με όλα τα tips που εισάγει ο χρήστης, η οποία θα χρησιμοποιηθεί για να σταλούν στη βάση δεδομένων. Ο λόγος που αποθηκεύονται εδώ είναι διότι το UploadTipsDialog class λειτουργεί σαν loop, ξεκινώντας ξανά και ξανά για την εισαγωγή κάθε tip, οπότε δεν θα ήταν ιδανικό για την αποθήκευση στατικών δεδομένων.

4.3.9. UploadTipsDialog.cs

Αυτός ο διάλογος λειτουργεί σαν loop. Εμφανίζει μία adaptive card ανά repetition και στη συνέχεια ξεκινά ξανά από την αρχή. Όταν ο αριθμός των tips που χρειάζεται να ανέβουν στη βάση εξαντληθεί, τα tips αποστέλλονται στη βάση δεδομένων προβάλλοντας το ανάλογο μήνυμα στο χρήστη και ο διάλογος τερματίζεται, επιστρέφοντας πίσω στο MainDialog.

```

var result = JsonConvert.DeserializeObject<Tip>(stepContext.Result.ToString());
NumberOfTipsDialog.TipsUploadList.Add(NullifyFalseValues(result));

if (--NumberOfTipsDialog.NumberOfTips == 0)
{
    // Send to database
    MainDialog.SendTipsToDB(NumberOfTipsDialog.TipsUploadList, stepContext, cancellationToken);

    // Show uploading message to the user
    var MsgText = MainDialog.Response.UploadingDataMessage();
    var promptMessage = MessageFactory.Text(MsgText, MsgText, InputHints.ExpectingInput);
    await stepContext.PromptAsync(nameof(TextPrompt), new PromptOptions { Prompt = promptMessage }, cancellationToken);

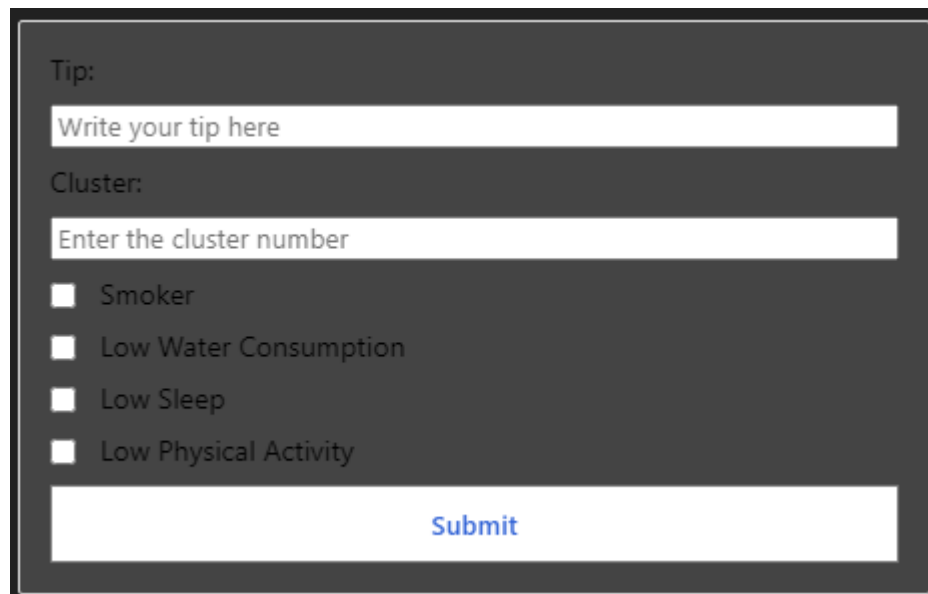
    return await stepContext.EndDialogAsync(PersonalDetailsDialog.PersonalDetails, cancellationToken);
}
else
{
    return await stepContext.BeginDialogAsync(nameof(UploadTipsDialog), NumberOfTipsDialog.NumberOfTips, cancellationToken);
}

```

Σχήμα 51: Μηχανισμός Loop για Εμφάνιση των Καρτών Εισαγωγής Tips

Για την εισαγωγή των tips γίνεται χρήση adaptive cards τα οποία και πάλι έχουν παραμετροποιηθεί για να λειτουργούν σαν διάλογος με τη χρήση του AdaptiveCardPrompt class.

Η περιγραφή της adaptive card γίνεται στο αρχείο tipCard.json. Περιέχει στο πάνω μέρος ένα TextBlock και ένα Input.Text όπου ο χρήστης εισάγει το κείμενο του tip. Από κάτω περιέχει άλλο ένα TextBlock και ένα Input.Text στο οποίο ο χρήστης καλείτε να δώσει τον αριθμό του cluster στο οποίο ανήκει το tip του. Στη συνέχεια παρατίθενται τέσσερα Input.Toggle τα οποία αφήνουν τον χρήστη να συμπληρώσει αν το tip απευθύνετε σε καπνιστές, άτομα με χαμηλή κατανάλωση νερού, άτομα με αυπνία και άτομα με χαμηλή φυσική άσκηση. Τέλος υπάρχει το Action.Submit κουμπί το οποίο πατά ο χρήστης μόλις τελειώσει την εισαγωγή του tip. Η εισαγωγή κειμένου για το tip είναι απαραίτητη, αλλά όλες οι υπόλοιπες τιμές είναι προαιρετικές και μπορούν να παραμείνουν κενές.



Σχήμα 52: Κάρτα Εισαγωγής Tip στον Emulator

Η συνάρτηση NullifyFalseValues μετατρέπει τις false τιμές που περιέχονται σε ένα tip σε null. Αυτό είναι ιδιαίτερα χρήσιμο όταν το rules engine επιλέγει ποια tips είναι κατάλληλα για κάθε χρήστη. Για παράδειγμα, ένας χρήστης που είναι καπνιστής θα πρέπει να βλέπει τα tips του cluster του τα οποία δεν λαμβάνουν υπόψη αν κάποιος είναι καπνιστής ή όχι. Παρόλα αυτά αποθηκεύοντας εκείνο το tip με τιμή Smoker == false σημαίνει ότι απευθύνετε μόνο σε μη καπνιστές, το οποίο δεν είναι απαραίτητα ορθό. Για αυτό το λόγο, αν η τιμή του Smoker (και όλων των υπολοίπων Input.Toggle) δεν είναι true, παίρνει τη τιμή null.

4.3.10. NameOfQuestionnaireDialog.cs

Αυτός ο διάλογος συλλέγει από το χρήστη το όνομα του ερωτηματολογίου, καθώς και τον αριθμό των ερωτήσεων που εμπεριέχονται μέσα στο ερωτηματολόγιο το οποίο ο χρήστης επιθυμεί να ανεβάσει στη βάση δεδομένων. Μόλις συλλεχθούν αυτά τα δεδομένα ο χρήστης παραπέμπεται στο διάλογο UploadQuestionnairesDialog, όπου ξεκινά να καταγραφεί τις ερωτήσεις του καινούριου ερωτηματολογίου του.

Σε αυτό το διάλογο αποθηκεύεται το όνομα του ερωτηματολογίου, ο αριθμός των ερωτήσεων, καθώς και το ίδιο το ερωτηματολόγιο το οποίο αποστέλλεται στη βάση δεδομένων.

4.3.11. UploadQuestionnairesDialog.cs

Ο τρόπος λειτουργίας αυτού του διαλόγου είναι παρόμοιος με τον UploadTipsDialog. Παρουσιάζεται ένα adaptive card στο οποίο ο χρήστης συμπληρώνει μία ερώτηση. Μέχρι ο αριθμός των ερωτήσεων να μηδενιστεί ο διάλογος ξεκινά από την αρχή για να εισαχθεί η επόμενη ερώτηση. Όταν όλες οι ερωτήσεις έχουν εισαχθεί, το ερωτηματολόγιο αποστέλλεται στη βάση δεδομένων και ο διάλογος τερματίζεται, επιστρέφοντας στο MainDialog.

```
var result = JsonConvert.DeserializeObject<QuestionJSON>(stepContext.Result.ToString());
NameOfQuestionnaireDialog.QuestionnaireToUpload.Add(ConvertToQuestionTopFive(result));

if (--NameOfQuestionnaireDialog.NumberOfQuestionsInQuestionnaire == 0)
{
    // Send to database
    MainDialog.SendQuestionnairesToDB(NameOfQuestionnaireDialog.NameOfQuestionnaire, NameOfQuestionnaireDialog.QuestionnaireToUpload, stepContext, cancellationTokens);

    // Show uploading message to the user
    var MsgText = MainDialog.Response.UploadingDataMessage();
    var promptMessage = MessageFactory.Text(MsgText, MsgText, InputHints.ExpectingInput);
    await stepContext.PromptAsync(nameof(TextPrompt), new PromptOptions { Prompt = promptMessage }, cancellationTokens);

    return await stepContext.EndDialogAsync(PersonalDetailsDialog.PersonalDetails, cancellationTokens);
}
else
    return await stepContext.BeginDialogAsync(nameof(UploadQuestionnairesDialog), NameOfQuestionnaireDialog.NumberOfQuestionsInQuestionnaire, cancellationTokens);
```

Σχήμα 53: Μηχανισμός Loop για Εμφάνιση των Καρτών Εισαγωγής Ερωτηματολογίων

Το adaptive card και πάλι χρησιμοποιείτε σαν διάλογος με τη χρήση του AdaptiveCardPrompt class. Η κάρτα περιγράφεται στο αρχείο question.json. Στο πάνω μέρος υπάρχει ένα TextBlock και ένα Input.Text, στο οποίο ο χρήστης καταγράφει την ερώτησή του. Στη συνέχεια υπάρχει ένα TextBlock με πέντε Input.ChoiceSet, ένα για κάθε personality trait (Extraversion, Agreeableness, Conscientiousness, Neuroticism, Openness) και ο χρήστης καλείται να επιλέξει ένα από αυτά. Στη συνέχεια υπάρχει ένα Input.Toggle το οποίο ενεργοποιείτε από το χρήστη όταν η ερώτηση είναι αρνητικής λογικής. Συνεχίζει με ένα TextBlock και πέντε Input.Text τα οποία αντιστοιχούν στις πέντε απαντήσεις οι οποίες προβάλλονται, όταν η ερώτηση χρησιμοποιείτε από το bot. Αυτό δίνει τη δυνατότητα για εξατομικευμένες απαντήσεις σε κάθε ερώτηση. Επίσης δίνεται και η δυνατότητα αυτά τα Input.Text να παραμείνουν κενά, και το bot αυτόματα αντικαταστεί τις κενές απαντήσεις με τις default απαντήσεις από το QuestionTopFive class. Τέλος υπάρχει το Action.Submit κουμπί το οποίο καταχωρεί την ερώτηση στο σύστημα.

Σχήμα 54: Κάρτα Εισαγωγής Ερώτησης στον Emulator

Η συνάρτηση `ConvertToQuestionTopFive` είναι υπεύθυνη να μετατρέπει τα δεδομένα της κάρτας σε αντικείμενο τύπου `QuestionTopFive`, ούτως ώστε να μπορεί να χρησιμοποιηθεί από το bot. Επίσης όταν οι απαντήσεις σε κάποια ερώτηση είναι κενές, προσφέρει τις default απαντήσεις στη συγκεκριμένη ερώτηση, τις οποίες παίρνει από το `QuestionTopFive` class.

```
if ((input.Answer1 == "") && (input.Answer2 == "") && (input.Answer3 == "") && (input.Answer4 == "") && (input.Answer5 == ""))
    output = new QuestionTopFive(question, personalityTrait, reverseLogic);
else
    output = new QuestionTopFive(question, personalityTrait, input.Answer1, input.Answer2, input.Answer3, input.Answer4, input.Answer5, reverseLogic);
```

Σχήμα 55: Απόδοση των Default Απαντήσεων σε Ερώτηση

4.3.12. `CancelAndHelpDialog.cs`

Αυτός ο διάλογος είναι κατασκευασμένος να διακόπτει όπου χρειάζεται τους άλλους διαλόγους που βρίσκονται σε χρήση. Ο λόγος για να διακοπεί κάποιος διάλογος είναι, είτε διότι ο χρήστης ζητά βοήθεια, είτε διότι ο χρήστης θέλει να διακόψει το συγκεκριμένο διάλογο και να επιστρέψει στην αρχική ροή του bot. Αν ο χρήστης ζητήσει βοήθεια του εμφανίζεται το μήνυμα της βοήθειας και στη συνέχεια συνεχίζεται ο διάλογος ο οποίος είχε διακοπεί. Στη περίπτωση που ο χρήστης επιθυμεί να τερματίσει τον παρόν διάλογο τότε ο διάλογος τερματίζεται και το bot επιστρέφει στο `MainDialog` παρουσιάζοντας το κατάλληλο μήνυμα.

```

if (Regex.IsMatch(text, "exit", RegexOptions.IgnoreCase) || Regex.IsMatch(text, "cancel", RegexOptions.IgnoreCase) || Regex.IsMatch(text, "quit", RegexOptions.IgnoreCase))
{
    var cancelMessage = MessageFactory.Text(CancelMsgText, CancelMsgText, InputHints.IgnoringInput);
    await innerDc.Context.SendActivityAsync(cancelMessage, cancellationToken);
    return await innerDc.CancelAllDialogsAsync(cancellationToken);
}

if (Regex.IsMatch(text, "help", RegexOptions.IgnoreCase))
{
    var helpMessageText = MainDialog.Response.HelpMessage();
    var helpMessage = MessageFactory.Text(helpMessageText, helpMessageText, InputHints.ExpectingInput);
    await innerDc.Context.SendActivityAsync(helpMessage, cancellationToken);
    return new DialogTurnResult(DialogTurnStatus.Waiting);
}

```

Σχήμα 56: Λειτουργία Βοήθειας και Εξόδου από Διάλογο

Μια σημαντική σημείωση είναι ότι αυτό το interrupt κάνει parse το input που δίνει ο χρήστης κάθε φορά. Λόγο του γεγονότος ότι ο χρήστης δε μπορεί υπό κανονικές συνθήκες να προχωρήσει στο διάλογο χωρίς να γράψει, ή να πει, κάτι στο bot, το συγκεκριμένο interrupt δεν υποστηρίζει null input. Παρόλα αυτά, με το custom prompt που δημιουργήθηκε για τη χρήση adaptive cards σε διάλογο, ο χρήστης μπορεί να προχωρήσει στο διάλογο γράφοντας τα στοιχεία του στη κάρτα και πατώντας ένα κουμπί από τη κάρτα. Αυτό αφήνει το input text πεδίο του χρήστη άδειο, το οποίο δημιουργεί προβλήματα στο interrupt του CancelAndHelpDialog class. Οπότε, για να αντιμετωπιστεί αυτό, προστέθηκε ένα check για non null τιμές στο input.

```

1 reference | demetrisbakas, 45 days ago | 1 author, 2 changes | 0 exceptions
private async Task<DialogTurnResult> InterruptAsync(DialogContext innerDc, CancellationToken cancellationToken)
{
    if (innerDc.Context.Activity.Type == ActivityTypes.Message && innerDc.Context.Activity.Text != null)
    {

```

Σχήμα 57: Έλεγχος για NULL Κείμενο Εισόδου

4.3.13. PersonalDetails.cs

Αυτό το class εμπεριέχει όλα τα προσωπικά στοιχεία του χρήστη. Δημιουργείτε ένα αντικείμενο αυτού του class στο PersonalDetailsDialog και κρατάτε συνεχώς ενήμερο με τη βάση δεδομένων. Όταν ο χρήστης προσθέσει ή αλλάξει κάτι στο αντικείμενο αυτό, αποστέλλεται στη βάση και κάθε φορά που ξεκινά κάποιος χρήστης συζήτηση με το bot κατεβαίνει από τη βάση.

Οι μεταβλητές που εμπεριέχει είναι:

Το *UserID*, το οποίο δημιουργείτε από το κανάλι επικοινωνίας και είναι μοναδικό για κάθε χρήστη.

Το *Name*, το οποίο απεικονίζει το όνομα του χρήστη και στις πλείστες περιπτώσεις παρέχεται από το κανάλι επικοινωνίας.

Το *Age*, που αποθηκεύει την ηλικία του χρήστη.

Το *Sex*, που κρατά το φύλο του χρήστη.

Το *Smoker*, το οποίο δείχνει αν ο χρήστης είναι καπνιστής ή όχι.

Το *WaterConsumption*, που περιέχει τον αριθμό των ποτηριών νερού που καταναλώνει ο χρήστης την ημέρα.

Το *Sleep*, που αποθηκεύει τις ώρες που κοιμάται ο χρήστης μέσα σε μία μέρα.

Το *PhysicalActivity*, το οποίο καταγράφει τις ώρες φυσικής άσκησης που πραγματοποιεί ο χρήστης μέσα σε μια βδομάδα.

Τα *Extraversion*, *Agreeableness*, *Conscientiousness*, *Neuroticism*, *Openness*, τα οποία κρατούν το αριθμητικό score του χρήστη για κάθε personality trait.

Το *Cluster*, το οποίο περιέχει τον αριθμό του cluster στο οποίο βρίσκετε ο χρήστης.

Τέλος το *QuestionnaireAnswers*, το οποίο είναι τύπου *IDictionary<string, int>* και αποθηκεύει κάθε απαντημένη ερώτηση που έχει ολοκληρώσει ο χρήστης, μαζί με την απάντηση που έχει δώσει. Αυτό χρησιμοποιείτε για να γνωρίζει το bot ποιες ερωτήσεις έχουν απαντηθεί από το χρήστη, καθώς και για να μπορεί να υπολογίσει το score στα personality traits.

Όλες οι μεταβλητές υποστηρίζουν τιμή null για να μπορούν να γίνονται οι κατάλληλοι έλεγχοι και να εξετάζεται αν ο χρήστης δεν έχει δώσει ακόμη τα στοιχεία του.

4.3.14. ResponseText.cs

Αυτό το class είναι εξαιρετικά σημαντικό για τη λειτουργία του bot καθώς περιέχει τα πλείστα μηνύματα τα οποία αποστέλλονται στο χρήστη από το bot. Μόλις το bot ξεκινά να τρέχει, ένα αντικείμενο αυτού του class δημιουργείτε στη *MainDialog* και από εκεί αντλούνται όλα τα μηνύματα.

Για κάθε μήνυμα το οποίο διαθέτει, υπάρχει αντίστοιχο function το οποίο είναι αρμόδιο να απαντήσει στο συγκεκριμένο ερώτημα. Τα μηνύματα εμπεριέχονται σε λίστες και το bot επιλέγει ένα μήνυμα τυχαία για να παρουσιάσει στο χρήστη. Αυτό κάνει το bot να φαίνεται πιο versatile καθώς δεν απαντά τις ίδιες ερωτήσεις με τον ίδιο τρόπο κάθε φορά. Για παράδειγμα υπάρχουν διάφοροι τρόποι με τους οποίους το bot θα χαιρετήσει ένα χρήστη, οι τρεις χαιρετισμοί είναι: "Hello!", "Hi!", "Hey!". Παρόλα αυτά, αν το bot γνωρίζει ήδη το όνομα του χρήστη, θα το χρησιμοποιήσει στο χαιρετισμό. Όλα αυτά πραγματοποιούνται από τη *ResponseText*, εμείς το μόνο που έχουμε να κάνουμε για να πάρουμε το string του χαιρετισμού είναι να καλέσουμε τη συνάρτηση *MainDialog.Response.Greet()* για χαιρετισμό χωρίς όνομα, και *MainDialog.Response.Greet(string name)* για χαιρετισμό με το όνομα του χρήστη.

```
// Response Lists
private readonly List<string> greetList = new List<string>() { "Hello!", "Hi!", "Hey!" };
```

Σχήμα 58: Λίστα Μηνυμάτων Χαιρετισμού

```
2 references | 0 changes | 0 authors, 0 changes | 0 exceptions
public string Greet()
{
    return RandomiseList(greetList);
}

1 reference | 0 changes | 0 authors, 0 changes | 0 exceptions
public string Greet(string name)
{
    return $"{Regex.Replace(Greet(), @"[^\\w\\s]", "")}, {name}!";
}
```

Σχήμα 59: Δημιουργία Μηνύματος Χαιρετισμού

Οι λόγοι που τα πλείστα μηνύματα επικοινωνίας του bot βρίσκονται σε αυτό το class είναι δύο. Ο πρώτος λόγος είναι για καλύτερη οργάνωση. Αν χρειαστεί να γίνουν αλλαγές στο κείμενο του bot οι αλλαγές γίνονται εύκολα και γρήγορα σε αυτό το class και χωρίς να χρειάζεται να έχουμε μεγάλα strings από μηνύματα μέσα στα αρχεία των διαλόγων. Ο δεύτερος λόγος είναι για να κάνει το bot να φαίνεται εξυπνότερο, καθώς τα requests του χρήστη μπορούν να απαντηθούν με διάφορους τρόπους.

Επίσης σε αυτή τη κλάση εμπεριέχεται η εκτέλεση του rules engine, η οποία ξεκινά όταν κληθεί η συνάρτηση `TipMessageAsync()`, η οποία είναι υπεύθυνη να επιστρέψει το string ενός tip το οποίο είναι κατάλληλο για το συγκεκριμένο χρήστη. Στην αρχή δημιουργείτε ένα rule repository στο οποίο φορτώνονται όλα τα rules. Μετά γίνεται το compilation και δημιουργείτε το session στο οποίο εφαρμόζονται οι κανόνες. Στη συνέχεια προστίθενται τα δεδομένα μέσα στο session. Στη προκειμένη περίπτωση, δεδομένα καθιστούν τα προσωπικά δεδομένα του χρήστη, καθώς και όλα τα tips που βρίσκονται στη βάση δεδομένων. Τέλος με τη κλήση της συνάρτησης `session.Fire()` ξεκινά το cycle το οποίο φιλτράρει τα tips τα οποία είναι κατάλληλα για το συγκεκριμένο χρήστη. Μόλις το cycle τελειώσει, απομένει μια λίστα με tips τα οποία είναι εφαρμόσιμα και ένα από αυτά επιλέγετε τυχαία για να δοθεί στο χρήστη.

```
2 references | demetrisbakas, 119 days ago | 1 author, 5 changes | 0 exceptions
public async Task<string> TipMessageAsync()
{
    TipList = new List<Tip>();
    var tipList = await MainDialog.QueryTipsAsync();

    // Rules engine here
    // Load rules
    var repository = new RuleRepository();
    repository.Load(x => x.From(typeof(TipChooseRule).Assembly));

    // Compile rules
    var factory = repository.Compile();

    // Create a working session
    var session = factory.CreateSession();

    // Insert facts into rules engine's memory
    session.Insert(PersonalDetailsDialog.PersonalDetails);
    foreach (Tip obj in tipList)
    {
        session.Insert(obj);
    }

    // Start match/resolve/act cycle
    session.Fire();

    return RandomiseList(TipList.Select(l => l.TipMessage).ToList());
}
```

Σχήμα 60: Λειτουργία Rules Engine

4.3.15. QuestionTopFive.cs

Από αυτή την κλάση δημιουργούνται τα αντικείμενα τα οποία απεικονίζουν ερωτήσεις στα ερωτηματολόγια του bot. Εμπεριέχει ένα αλφαριθμητικό με την ερώτηση, μία λίστα

με απαντήσεις, ένα enum το οποίο προσδιορίζει το personality trait στο οποίο αναφέρεται η ερώτηση και μια μεταβλητή bool η οποία καθορίζει αν η ερώτηση είναι αρνητικής λογικής.

Παρέχονται πολλαπλοί δημιουργοί (constructors) για τη κλάση ούτως ώστε να μπορούν εύκολα και γρήγορα να δημιουργηθούν ερωτήσεις. Για παράδειγμα, διαφορετικός constructor για αντικείμενα που είναι αρνητικής λογικής και διαφορετικοί constructors για να δίνονται custom απαντήσεις, ή για να χρησιμοποιούνται οι default.

Η συνάρτηση DefaultAnswers είναι υπεύθυνη να προσφέρει τις default απαντήσεις, νοουμένου ότι η δεδομένη ερώτηση δεν έχει δικές τις εξατομικευμένες απαντήσεις.

4.3.16. Tip.cs

Η χρήση της κλάσης Tip είναι να δημιουργούνται αντικείμενα από αυτή, τα οποία θα απεικονίζουν ένα tip το καθένα. Εμπεριέχει ένα string με το μήνυμα του tip, ένα int με το cluster στο οποίο ανήκει το συγκεκριμένο tip και τέσσερις bool μεταβλητές που δείχνουν αν το tip είναι κατάλληλο για καπνιστές, άτομα με χαμηλή κατανάλωση νερού, άτομα με λίγες ώρες ύπνου και άτομα που κάνουν λίγη φυσική άσκηση. Όλες οι μεταβλητές έχουν τη δυνατότητα να είναι null, σε περίπτωση που ο χρήστης δεν έχει συμπληρώσει ακόμη τα προσωπικά του στοιχεία.

4.3.17. TipChooseRule.cs

Αυτή η κλάση εμπεριέχει τα rules τα οποία καθορίζουν ποια tips είναι κατάλληλα για τον κάθε χρήστη. Σε περίπτωση που χρειαστεί να προστεθούν καινούρια rules, μπορούν, είτε να προστεθούν σε αυτό το αρχείο, είτε να δημιουργηθεί κι άλλο class το οποίο κάνει inherit την κλάση Rule. Και με τους δύο τρόπους δε θα χρειαστούν περεταίρω παραμετροποιήσεις για να δουλέψουν τα καινούρια rules, καθώς θα γίνουν compile αυτόματα από το rules engine.

Σε αυτό το αρχείο αποθηκεύονται επίσης, τα κατώφλια τα οποία καθορίζουν αν ο συγκεκριμένος χρήστης πίνει λίγο νερό, κοιμάται λίγο ή κάνει λίγη φυσική άσκηση.

Για να εφαρμοστούν τα rules, φορτώνονται στη μνήμη του rules engine τα προσωπικά δεδομένα του χρήστη και μια λίστα με όλα τα tips από τη βάση δεδομένων. Αρχικά συγκρίνεται το cluster του tip με το cluster του χρήστη. Αν το cluster είναι το ίδιο, ή αν η τιμή του cluster στο tip είναι null, τότε συνεχίζεται η σύγκριση, αυτή τη φορά με τη μεταβλητή Smoker. Αν ο χρήστης και το tip έχουν την ίδια τιμή στη μεταβλητή Smoker, ή αν η τιμή της μεταβλητής Smoker στο tip είναι null, τότε και πάλι η σύγκριση συνεχίζεται. Αν η τιμή ύπνου του χρήστη είναι μικρότερη ή ίση με το φράγμα, και η μεταβλητή LowSleep του tip έχει τιμή true, ή αν η τιμή του LowSleep είναι null, τότε και πάλι συνεχίζεται η σύγκριση. Αν η τιμή της φυσικής άσκησης του χρήστη είναι μικρότερη ή ίση από το φράγμα και η τιμή LowPhysicalActivity του tip είναι true, ή αν η τιμή του LowPhysicalActivity του tip είναι null, τότε και πάλι συνεχίζεται η σύγκριση. Τέλος, αν η τιμή νερού του χρήστη είναι μικρότερη ή ίση του φράγματος και η τιμή LowWaterConsumption του tip είναι true, ή αν η τιμή του LowWaterConsumption του tip είναι null, τότε το tip έχει περάσει επιτυχώς όλους τους κανόνες και είναι κατάλληλο για το συγκεκριμένο χρήστη, οπότε καλείτε η συνάρτηση AddTips(tips). Η συνάρτηση AddTips(tips) έχει ως αρμοδιότητα να προσθέτει όλα τα tips τα οποία είναι κατάλληλα για το συγκεκριμένο χρήστη σε μια λίστα, από την οποία η συνάρτηση MainDialog.Response.TipMessageAsync() θα επιλέξει ένα tip τυχαία.

```

PersonalDetails personalDetails = default;
IEnumerable<Tip> tips = default;

When()
.Match<PersonalDetails>(() => personalDetails)
// Cluster rule
.Query(() => tips, x => x.Match<Tip>(o => o.Cluster == personalDetails.Cluster || o.Cluster == null,
// Smoker rule
o => o.Smoker == personalDetails.Smoker || o.Smoker == null,
// Low sleep rule
o => (o.LowSleep == true && personalDetails.Sleep <= lowSleepThreshold) || o.LowSleep == null,
// Low physical activity rule
o => (o.LowPhysicalActivity == true && personalDetails.PhysicalActivity <= lowPhysicalActivityThreshold) || o.LowPhysicalActivity == null,
// Low water consumption rule
o => (o.LowWaterConsumption == true && personalDetails.WaterConsumption <= lowWaterConsumption) || o.LowWaterConsumption == null)
.Collect()
.Where(c => c.Any());

Then()
.Do(ctx => AddTips(tips));

```

Σχήμα 61: Κανόνες για το Rules Engine

4.3.18. NotifyController.cs

Η κλάση NotifyController είναι υπεύθυνη να χειρίζεται τα proactive messages. Δηλαδή, κάθε φορά που θα πραγματοποιηθεί ένα get request στο proactive messages endpoint του bot, αυτή η κλάση πρέπει να στείλει σε κάθε χρήστη ένα tip. Χρησιμοποιείτε ένα ConcurrentDictionary το οποίο περιέχει όλα τα conversation references των χρηστών. Το conversation reference του κάθε χρήστη δημιουργείται μόλις ο χρήστης στείλει ένα μήνυμα στο bot και αποθηκεύεται στο ConcurrentDictionary<string, ConversationReference> _conversationReferences. Από εκεί, για κάθε χρήστη του οποίου το conversation reference υπάρχει στο ConcurrentDictionary, καλείτε η συνάρτηση BotCallback. Επίσης υπάρχει ένα αντικείμενο ContentResult το οποίο απαντά στο get request που γίνεται στο endpoint, με μία html σελίδα που γράφει "Proactive messages have been sent."

```

0 references | demetrisbakas, 21 days ago | 1 author, 10 changes | 0 requests | 0 exceptions
public async Task<IActionResult> Get()
{
    try
    {
        foreach (var conversationReference in _conversationReferences.Values)
        {
            await ((BotAdapter)_adapter).ContinueConversationAsync(_appId, conversationReference, BotCallback, default);
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
    }

    // Let the caller know proactive messages have been sent
    return new ContentResult()
    {
        Content = "<html><body><h1>Proactive messages have been sent.</h1></body></html>",
        ContentType = "text/html",
        StatusCode = (int)HttpStatusCode.OK,
    };
}

```

Σχήμα 62: Αποστολή Μηνύματος (Tip) σε κάθε Conversation Reference

Η συνάρτηση BotCallback ελέγχει αν ο χρήστης έχει καταταχθεί σε cluster, που σημαίνει ότι έχει ολοκληρώσει τουλάχιστον ένα ερωτηματολόγιο και στη συνέχεια καλεί τη συνάρτηση MainDialog.Response.TipMessageAsync(), η οποία επιστρέφει ένα κατάλληλο tip για το συγκεκριμένο χρήστη, το οποίο μετά του παρουσιάζεται από το bot σε μορφή μηνύματος. Αν ο χρήστης εκείνη τη στιγμή βρίσκεται σε κάποιο άλλο διάλογο, τότε αφότου εμφανιστεί το tip, επιστρέφει στο διάλογο που βρισκόταν πριν.

```

1 reference | demetrisbakas, 21 days ago | 1 author, 11 changes | 0 exceptions
private async Task BotCallback(ITurnContext turnContext, CancellationToken cancellationToken)
{
    // http://localhost:3978/api/notify
    if (PersonalDetailsDialog.PersonalDetails.Cluster != null)
    {
        var tipMessage = await MainDialog.Response.TipMessageAsync();
        await turnContext.SendActivityAsync(tipMessage);
    }
}

```

Σχήμα 63: Επιλογή Μηνύματος (Tip) και Απόδοσή του στο Χρήστη

4.3.19. DialogBot.cs

Αυτή η κλάση χειρίζεται κάποια σημαντικά κομμάτια του διαλόγου στο bot. Πιο συγκεκριμένα παρέχει συναρτήσεις οι οποίες καλούνται από το bot κατά τη διάρκεια του διαλόγου και καλύπτουν διάφορες λειτουργίες. Εδώ, επίσης, αποθηκεύεται το ConcurrentDictionary με τα conversation references όλων των χρηστών.

Η συνάρτηση OnTurnAsync, καλείτε σε κάθε γύρο λειτουργίας του bot ανεξάρτητα αν μιλά ο χρήστης ή το ίδιο το bot. Μέσα από αυτή τη συνάρτηση συλλέγονται, το ID και το όνομα του χρήστη, από το κανάλι επικοινωνίας και καλούνται τα δεδομένα του χρήστη, από τη βάση δεδομένων, καθώς και τα ερωτηματολόγια και τα δεδομένα που χρειάζονται για το clustering, τα οποία βρίσκονται επίσης στη βάση δεδομένων. Ο λόγος που αυτές οι λειτουργίες εκτελούνται μέσω αυτής της συνάρτησης, είναι διότι η συνάρτηση καλείτε κατευθείαν όταν ο χρήστης ξεκινήσει μία συζήτηση με το bot και καλείται και κάθε επόμενο turn της συζήτησής τους. Αυτό παρέχει το χρονικό περιθώριο στα δεδομένα να αντληθούν από τη βάση δεδομένων πριν ο χρήστης να τα χρειαστεί. Αν ο χρήστης χρειαστεί κάποια από τα δεδομένα πριν αυτά προλάβουν να αντληθούν από τη βάση δεδομένων τότε θα περιμένει εκεί, μέχρι τα δεδομένα να φθάσουν. Αυτό επιτυγχάνεται με τη χρήση Task τα οποία ξεκινούν να αντλούν τα δεδομένα μόλις κληθεί για πρώτη φορά η συνάρτηση OnTurnAsync, και όταν ο χρήστης χρειάζεται να χρησιμοποιήσει τα δεδομένα, η χρήση της await τον αναγκάζει να περιμένει μέχρι αυτά να φθάσουν πλήρως.

```

0 references | demetrisbakas, 274 days ago | 1 author, 2 changes | 0 exceptions
public override async Task OnTurnAsync(ITurnContext turnContext, CancellationToken cancellationToken = default(CancellationToken))
{
    if (PersonalDetailsDialog.PersonalDetails?.UserID == null || turnContext.Activity.From.Id != PersonalDetailsDialog.PersonalDetails?.UserID)
    {
        PersonalDetailsDialog.PersonalDetails = new PersonalDetails();
        PersonalDetailsDialog.PersonalDetails.UserID = turnContext.Activity.From.Id;
        PersonalDetailsDialog.PersonalDetails.Name = turnContext.Activity.From.Name;

        MainDialog.ReadFromDb = MainDialog.CosmosDBQuery.ReadAsync(new string[] { PersonalDetailsDialog.PersonalDetails.UserID }, cancellationToken);
        MainDialog.ClusteringData = MainDialog.QueryClusterDetailsAsync();
        MainDialog.Questionnaires = MainDialog.QueryQuestionnairesAsync();
    }

    await base.OnTurnAsync(turnContext, cancellationToken);

    // Save any state changes that might have occurred during the turn.
    await ConversationState.SaveChangesAsync(turnContext, false, cancellationToken);
    await UserState.SaveChangesAsync(turnContext, false, cancellationToken);
}

```

Σχήμα 64: Συλλογή Δεδομένων Χρήστη από το Κανάλι και Εκκίνηση Tasks

Η συνάρτηση OnMessageActivityAsync καλείτε κάθε φορά που ο χρήστης στέλνει μήνυμα στο bot. Μέσα σε αυτή τη συνάρτηση συλλέγεται το conversation reference του χρήστη, με τη κλήση της συνάρτησης AddConversationReference.

Η συνάρτηση `AddConversationReference` είναι υπεύθυνη να καταχωρεί το `conversation reference` του χρήστη μέσα στο `ConcurrentDictionary` που περιέχονται όλα τα `conversation references` με τις ανοιχτές συζητήσεις του bot.

```
1 reference | demetrisbakas, 22 days ago | 1 author, 2 changes | 0 exceptions
protected static void AddConversationReference(Activity activity)
{
    var conversationReference = activity.GetConversationReference();
    _conversationReferences.AddOrUpdate(conversationReference.User.Id, conversationReference, (key, newValue) => conversationReference);
}
```

Σχήμα 65: Καταχώριση του Conversation Reference του Χρήστη

4.3.20. AdaptiveCardPrompt.cs

Αυτό το class έχει κατασκευαστεί ούτως ώστε να μπορεί το bot να χειρίζεται τα `adaptive cards` ως διάλογο. Υπό κανονικές συνθήκες το bot χειρίζεται τα `adaptive cards` σε turns. Παρόλα αυτά όταν ο χρήστης ζητείται να συμπληρώσει δεδομένα στο `adaptive card`, τότε πρέπει να πατά ένα κουμπί (το `Submit`) και το bot να συνεχίζει το διάλογο. Όποτε η χρήση του `adaptive card` σε turn δεν είναι πια εφικτή. Το `adaptive card` πρέπει να χρησιμοποιείται σαν `prompt`. Αυτό το class επιτρέπει στο `adaptive card` να χρησιμοποιηθεί σαν `prompt` και όταν ο χρήστης απαντήσει πατώντας το `Submit`, να συνεχίζεται η ροή του διαλόγου.

4.3.21. PrepareCard.cs

Αυτό το class χρησιμοποιείται για να διαβάζει τα δεδομένα που ο χρήστης συμπληρώνει σε ένα `adaptive card` καλώντας την συνάρτηση `PrepareCard.ReadCard` η οποία φτιάχνει το `path` και διαβάζει το κείμενο που εισάχθηκε μέσα στην κάρτα.

```
3 references | demetrisbakas, 77 days ago | 1 author, 1 change | 0 exceptions
public static string ReadCard(string fileName)
{
    string[] BuildPath = { ".", "Cards", fileName };
    var filePath = Path.Combine(BuildPath);
    var fileRead = File.ReadAllText(filePath);
    return fileRead;
}
```

Σχήμα 66: Διάβασμα JSON από Adaptive Card

4.3.22. LuisModel.cs

Σε αυτή την κλάση εφάπτεται το αποτέλεσμα που παίρνουμε σε μορφή `JSON` όταν κάνουμε request στο `LUIS`. Περιέχει όλα τα `intents` και `entities` που χρησιμοποιούνται από το `LUIS` μοντέλο του bot, τα οποία περιέχονται επίσης και στο `LuisModel.json`.

```

8 references | demetrisbakas, 17 hours ago | 1 author, 3 changes
public enum Intent {
    AddQuestionnairesOrTips,
    AnswerQuestionnaires,
    EnterPersonalDetails,
    Greet,
    None
};
public Dictionary<Intent, IntentScore> Intents;

```

Σχήμα 67: Διαθέσιμα Intents

```

// Built-in entities
public string[] personName;
public Age[] age;

```

Σχήμα 68: Built-in Entities

4.3.23. Startup.cs

Αυτή η κλάση τρέχει με την έναρξη του bot και αρχικοποιεί όλα τα services που χρησιμοποιούνται από το bot. Εδώ υπάρχουν επίσης όλοι οι διάλογοι που χρησιμοποιούνται, καθώς και το ConcurrentDictionary για τα conversation references.

4.3.24. ConnectionRecognizer.cs

Αυτό το class είναι υπεύθυνο για τη σύνδεση του LUIS και του QnA Maker με το bot. Για το LUIS χρησιμοποιείτε το μοντέλο που δημιουργήθηκε για το bot από το Azure. Για το QnA Maker χρησιμοποιείτε το ActivityCoachingBotKnowledgeBase ως knowledge base το οποίο περιέχει το chit-chat module. Με τη κλήση αυτού του recognizer μπορούμε να παραλάβουμε τα αποτελέσματα από το LUIS καθώς και το knowledge base. Για τη σύνδεση του LUIS και του QnA Maker χρησιμοποιούνται τα αντίστοιχα NuGet packages. Ο λόγος που το set στο function είναι private, είναι διότι χρειάζεται μόνο να λάβουμε την απάντηση από το knowledge base, καθώς το utterance του χρήστη αποστέλλεται από τη συνάρτηση RecognizeAsync.

```

SampleQnA = new QnAMaker(new QnAMakerEndpoint
{
    KnowledgeBaseId = QnAKnowledgebaseId,
    EndpointKey = QnAEndpointKey,
    Host = QnAEndpointHostName
});

2 references | demetrisbakas, 128 days ago | 1 author, 1 change | 0 exceptions
public QnAMaker SampleQnA { get; private set; }

```

Σχήμα 69: Σύνδεση QnA Maker

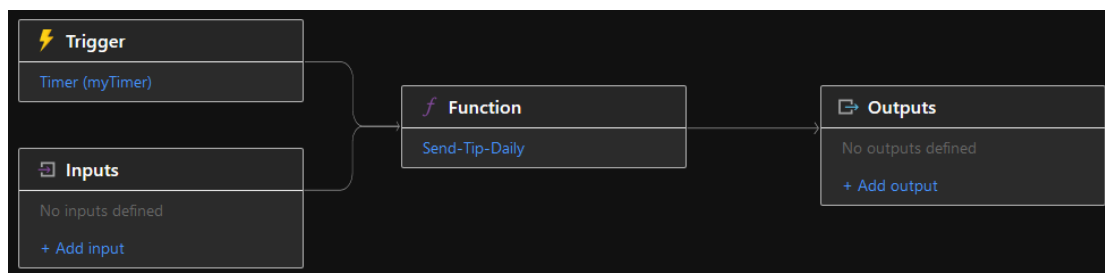
4.3.25. NuGet Packages

Εδώ παρατάσσονται όλα τα πακέτα κώδικα που χρησιμοποιήθηκαν στο solution.

- **AdaptiveCards** (by Microsoft) v2.1.0
- **Microsoft.AspNetCore.App** (by Microsoft) v2.1.1
- **Microsoft.Bot.Builder.AI.Luis** (by Microsoft) v4.8.0
- **Microsoft.Bot.Builder.AI.QnA** (by Microsoft) v4.9.1
- **Microsoft.Bot.Builder.Azure** (by Microsoft) v4.7.3
- **Microsoft.Bot.Builder.Dialogs** (by Microsoft) v4.9.1
- **Microsoft.Bot.Builder.Integration.AspNet.Core** (by Microsoft) v4.8.2
- **Microsoft.Bot.Connector.Teams** (by Microsoft) v0.10.0
- **Microsoft.ML** (by Microsoft) v1.5.0
- **Microsoft.NETCore.App** (by Microsoft) v2.1.0
- **Microsoft.Recognizers.Text.DataTypes.TimexExpression** (by Microsoft) v1.3.2
- **NRules** (by Sergiy Nikolayev) v0.9.1

4.3.26. Azure Function

Αυτό το azure function app, έχει στόχο να διαμοιράζει καθημερινά τα tips σε όλους τους χρήστες που έχουν ξεκινήσει συζήτηση με το bot. Διαθέτει ένα timer το οποίο χρησιμοποιείτε σαν trigger και ένα function, το Send-Tip-Daily, το οποίο ενεργοποιείται από το trigger. Το timer περιέχει ένα cron expression το οποίο υποδεικνύει ότι το trigger ενεργοποιείτε καθημερινά μία συγκεκριμένη ώρα. Όταν το trigger ενεργοποιηθεί, τρέχει το function.



Σχήμα 70: Azure Function Flowchart

Το function Send-Tip-Daily εκτελεί ένα get request στο proactive messages endpoint του bot. Έπειτα ο κώδικας του bot αναλαμβάνει να διαμοιράσει το κατάλληλο tip στον κάθε χρήστη.


```

public static async Task Run(TimerInfo myTimer, ILogger log)
{
    log.LogInformation($"C# Timer trigger function executed at: {DateTime.Now}");

    const string URL = "https://activitycoachingbot.azurewebsites.net/api/notify";

    HttpClient client = new HttpClient();
    client.BaseAddress = new Uri(URL);

    await client.GetAsync(URL);
}

```

Σχήμα 71: Get Request στο Endpoint από το Azure Function

4.3.27. Database – Azure Cosmos DB

Για τη βάση δεδομένων χρησιμοποιήθηκε το Azure Cosmos DB λόγο του χαμηλού latency και της ευκολίας του στη χρήση. Χρησιμοποιήθηκε το SQL API ούτως ώστε να είναι δυνατών να πραγματοποιηθούν queries όπως σε μία συνηθισμένη SQL βάση δεδομένων. Δημιουργήθηκαν τρία containers τα οποία εμπεριέχουν objects που στέλνονται ή παραλαμβάνονται από το bot σε μορφή JSON.

Το container με το όνομα bot-storage περιέχει τα προσωπικά δεδομένα των χρηστών και τα αποθηκεύει σε objects τύπου PersonalDetails.

Το container questionnaires περιέχει όλα τα ερωτηματολόγια του bot και τα αποθηκεύει σε objects τύπου KeyValuePair<string, List<QuestionTopFive>>. Το key του KeyValuePair περιέχει το όνομα του ερωτηματολογίου σε τύπο string και η λίστα με τα QuestionTopFive, είναι η λίστα που περιέχει τις ερωτήσεις του ερωτηματολογίου.

Το container tips περιέχει όλα τα tips που είναι διαθέσιμα στο bot και τα αποθηκεύει σε objects τύπου Tip.

4.3.28. LUIS (Language Understanding)

Το LUIS περιέχει το machine learning μοντέλο το οποίο αναλύει τον γραπτό λόγο και επιστρέφει στο bot ένα JSON stream με τις προθέσεις (intents) του χρήστη και διάφορες χρήσιμες μεταβλητές (entities) οι οποίες γίνονται parse από αυτά που ο χρήστης γράφει σε φυσική γλώσσα.

Τα intents υποδεικνύουν τις προθέσεις του χρήστη και είναι ο τρόπος με τον οποίο το bot αντιλαμβάνεται πιο διάλογο επιθυμεί να εκκινήσει ο χρήστης. Τα intents που δημιουργήθηκαν στο μοντέλο LUIS είναι τα εξής:

- **AddQuestionnairesOrTips.** Αυτό το intent υποδεικνύει την επιθυμία του χρήστη να προσθέσει ερωτηματολόγια ή tips στη βάση δεδομένων.
- **AnswerQuestionnaires.** Αυτό το intent υποδεικνύει την επιθυμία του χρήστη να απαντήσει ερωτηματολόγια.
- **EnterPersonalDetails.** Αυτό το intent υποδεικνύει την επιθυμία του χρήστη να συμπληρώσει, ή να ανανεώσει τα προσωπικά του δεδομένα.

- **Greet.** Αυτό το intent περιγράφει χαιρετισμό του χρήστη προς το bot.
- **None.** Σε αυτό το intent κατατάσσονται τα utterances τα οποία δε μπορούν να καταταχθούν σε κάποιο άλλο intent και δεν έχουν κάποια χρήση προς το bot.

Τα entities συμβολίζουν χρήσιμες μεταβλητές οι οποίες αντλούνται από τα λεγόμενα του χρήστη. Τα entities τα οποία το LUIS μοντέλο αναγνωρίζει είναι τα εξής:

- **age.** Prebuilt μεταβλητή η οποία παρέχεται από το LUIS και προσδιορίζει ηλικία. Χρήσιμη για όταν χρειάζεται να καταγραφεί η ηλικία του χρήστη.
- **personName.** Prebuilt μεταβλητή η οποία παρέχεται από το LUIS και προσδιορίζει όνομα. Χρήσιμη για όταν χρειάζεται να καταγραφεί το όνομα του χρήστη.
- **Sex.** Μεταβλητή τύπου λίστας (List) η οποία περιέχει τις λέξεις “Male” και “Female”. Χρήσιμη για όταν χρειάζεται να καταγραφεί το φύλο του χρήστη.

4.3.29. QnA Maker

Στον QnA Maker υπάρχουν συχνές ερωτήσεις που γίνονται σε bots μαζί με τις απαντήσεις τους. Όλα τα pairs ερωτήσεων-απαντήσεων παρέχονται από το Chat knowledge base το οποίο είναι ρυθμισμένο στο “Friendly” πακέτο ερωτήσεων. Περιέχει 90 πιθανές ερωτήσεις, μαζί με τις απαντήσεις που θα δώσει το bot όταν η κάθε ερώτηση προκύψει. Για κάθε pair υπάρχουν πολλαπλές διαφορετικές ερωτήσεις οι οποίες είναι συσχετισμένες στην ίδια απάντηση για να βοηθούν το μοντέλο να αντιλαμβάνεται καλύτερα αυτό που ρωτά ο χρήστης. Υπάρχει δυνατότητα να προστεθούν καινούρια pairs ερωτήσεων-απαντήσεων, καινούριες απαντήσεις και ερωτήσεις σε κάποιο ήδη υπάρχων pair, καθώς και δυνατότητα να προστεθούν καινούρια knowledge bases τα οποία θα λειτουργούν κατευθείαν με το bot.

5. Βιβλιογραφία

- [1] M. Rouse, «Definition IM bot,» Σεπτέμβριος 2005. [Ηλεκτρονικό]. Available: <https://searchdomino.techtarget.com/definition/IM-bot>.
- [2] J. Baron, «2017 Messenger Bot Landscape, a Public Spreadsheet Gathering 1000+ Messenger Bots,» 3 Μάιος 2017. [Ηλεκτρονικό]. Available: <https://cai.tools.sap/blog/2017-messenger-bot-landscape/>.
- [3] M. B. Hoy, «Alexa, Siri, Cortana, and More: An Introduction to Voice Assistants,» *EMERGING TECHNOLOGIES*, pp. 81-88, 12 Ιανουαρίου 2018.
- [4] A. Turing, «COMPUTING MACHINERY AND INTELLIGENCE,» 1950, pp. 433-460.
- [5] J. Weizenbaum, «ELIZA--A Computer Program For the Study of Natural Language Communication Between Man and Machine,» Massachusetts Institute of Technology, 1966.
- [6] B. Copeland, «Artificial intelligence,» Britannica, [Ηλεκτρονικό]. Available: <https://www.britannica.com/technology/artificial-intelligence/Methods-and-goals-in-AI>.
- [7] Enterprise Architecture, «AI Chatbot Reference Architecture,» 01 Ιανουαρίου 2019. [Ηλεκτρονικό]. Available: <https://www.dragon1.com/watch/470340/chatbot-reference-architecture>.
- [8] P. Christensson, «Framework Definition,» TechTerms, 07 Μαρτίου 2013. [Ηλεκτρονικό]. Available: <https://techterms.com/definition/framework>.
- [9] Google, «Dialogflow,» [Ηλεκτρονικό]. Available: <https://cloud.google.com/dialogflow/>.
- [10] IBM, «Watson Assistant v2,» 23 Σεπτεμβρίου 2020. [Ηλεκτρονικό]. Available: <https://cloud.ibm.com/apidocs/assistant/assistant-v2>.
- [11] Botpress, «botpress,» [Ηλεκτρονικό]. Available: <https://botpress.com/>.
- [12] Microsoft, «Azure Bot Service,» [Ηλεκτρονικό]. Available: <https://azure.microsoft.com/en-us/services/bot-service/>.
- [13] Microsoft, «What is the Bot Framework SDK?,» 15 Νοεμβρίου 2019. [Ηλεκτρονικό]. Available: <https://docs.microsoft.com/en-us/azure/bot-service/bot-service-overview-introduction?view=azure-bot-service-4.0>.
- [14] Microsoft, «microsoft / botframework-sdk,» GitHub, [Ηλεκτρονικό]. Available: <https://github.com/Microsoft/botframework-sdk>.
- [15] Google, «Clustering in Machine Learning,» 02 Οκτωβρίου 2020. [Ηλεκτρονικό]. Available: <https://developers.google.com/machine-learning/clustering/overview>.
- [16] Wikimedia Foundation, «Business rules engine,» pp. 1-2, 2012.
- [17] F. W. D. A. (. A. L. D. B. Hayes-Roth, Building expert systems, Reading, Massachusetts: Addison-Wesley Pub. Co., 1983.
- [18] Microsoft, «LUIS,» [Ηλεκτρονικό]. Available: <https://www.luis.ai/>.
- [19] Microsoft, «What is QnA Maker?,» [Ηλεκτρονικό]. Available: <https://docs.microsoft.com/en-us/azure/cognitive-services/QnAMaker/Overview/overview>.

- [20] Microsoft, «Welcome to Azure Cosmos DB,» [Ηλεκτρονικό]. Available: <https://docs.microsoft.com/en-us/azure/cosmos-db/introduction>.
- [21] Microsoft, «ML.NET,» [Ηλεκτρονικό]. Available: <https://dotnet.microsoft.com/apps/machinelearning-ai/ml-dotnet>.
- [22] EDUCBA, «K- Means Clustering Algorithm,» [Ηλεκτρονικό]. Available: <https://www.educba.com/k-means-clustering-algorithm/>.
- [23] S. Nikolayev, «NRules wiki,» [Ηλεκτρονικό]. Available: <https://github.com/NRules/NRules/wiki>.
- [24] W.-r. Jih, «Dwelling in the Dream (Rete Algorithm),» 22 March 2007. [Ηλεκτρονικό]. Available: <https://wrjih.wordpress.com/2007/03/22/rete-algorithm/>.
- [25] Microsoft, «Introduction to Azure Functions,» Microsoft, 20 11 2020. [Ηλεκτρονικό]. Available: <https://docs.microsoft.com/en-us/azure/azure-functions/functions-overview>.
- [26] Microsoft, «Adaptive Cards Overview,» Microsoft, 06 Ιούνιος 2017. [Ηλεκτρονικό]. Available: <https://docs.microsoft.com/en-us/adaptive-cards/>.