

Ανάπτυξη Διαλογικού Πράκτορα με Rasa Framework

Περιεχόμενα

1. Ανάλυση προβλήματος
2. Ανάλυση χρηστών και σενάρια χρήσης
3. Rasa framework
4. Υλοποίηση και Αποτελέσματα
5. Συμπεράσματα

1. Ανάλυση προβλήματος

Ο σκοπός της παρούσας εργασίας είναι να δημιουργήσουμε ένα διαλογικό πράκτορα (chatbot) που θα μπορεί να εξυπηρετεί τις ανάγκες των φοιτητών Τμήματος Πληροφορικής και να παρέχει απαντήσεις στα ερωτήματά τους.

Στο πλαίσιο αυτής της εργασίας, θα δούμε πώς μπορεί να γίνει ανάπτυξη ενός διαλογικού πράκτορα (chatbot) με το Rasa framework. Συγκεκριμένα, θα αναπτύξουμε ένα chatbot που θα είναι σε θέση να απαντήσει σε ερωτήσεις σχετικά με το Τμήμα Πληροφορικής του ΑΠΘ, λειτουργώντας σαν γραμματεία. Ωστόσο, για τις ερωτήσεις που δεν σχετίζονται με το Τμήμα Πληροφορικής, το chatbot θα λαμβάνει απαντήσεις από το ChatGPT μέσω API.

Στο πλαίσιο αυτής της εργασίας θα μελετήσουμε την διαδικασία ανάπτυξης ενός chatbot και θα δούμε τι δυνατότητες μπορεί να προσφέρει ενσωμάτωση του ChatGPT στους χρήστες.

2. Ανάλυση χρηστών και σενάρια χρήσης

Οι κύριοι χρήστες που θα χρησιμοποιούν το chatbot θα είναι φοιτητές του Τμήματος Πληροφορικής, οι οποίοι θα θέτουν ερωτήσεις σχετικά με το Τμήμα και θα λαμβάνουν σχετικές απαντήσεις.

Παρακάτω αναλύονται κάποιες από αυτές τις ερωτήσεις και οι αναμενόμενες απαντήσεις που θα πρέπει να δίνει το chatbot:

Γενικές Ερωτήσεις:

Ερώτηση: Γεια σας!

Απάντηση: Γεια σας! Πώς μπορώ να σας βοηθήσω;

Ερώτηση: Ευχαριστώ για τη βοήθεια.

Απάντηση: Αν έχετε ερωτήσεις, είμαι πάντα έτοιμος να σας βοηθήσω.

Ερώτηση: Αντίο.

Απάντηση: Καλή συνέχεια. Είμαι πάντα έτοιμος να σας βοηθήσω.

Ερώτηση: Ποιος είσαι;

Απάντηση: Είμαι SecretaryChatbot που παρέχει πληροφορίες σχετικά με το Τμήμα Πληροφορικής του ΑΠΘ.

Ειδικές Ερωτήσεις:

Ερώτηση: Πώς μπορώ να πάρω βεβαίωση σπουδών;

Απάντηση: Μπορείς να πάρεις βεβαίωση σπουδών εδώ:
<https://students.auth.gr/#/requests/list>

Ερώτηση: Ποια είναι η ιστοσελίδα του Τμήματος;

Απάντηση: Μπορείτε να βρείτε την ιστοσελίδα του Τμήματος στη διεύθυνση
<https://www.csd.auth.gr/>

Ερώτηση: Ποιο είναι το email της γραμματείας;

Απάντηση: Το email της γραμματείας του Τμήματος Πληροφορικής είναι
info@csd.auth.gr

Ερώτηση: Ποιο είναι το τηλέφωνο της γραμματείας;

Απάντηση: Το τηλέφωνο της γραμματείας είναι 2310-99-8420

Ερώτηση: Ποια είναι η διεύθυνση της γραμματείας;

Απάντηση: Το Τμήμα Πληροφορικής διαθέτει εγκαταστάσεις τόσο στο κέντρο της πόλης, εντός της Πανεπιστημιούπολης του ΑΠΘ, όσο και στην Καλαμαριά. Η διεύθυνση γραμματείας είναι:

Γραμματεία Τμήματος Πληροφορικής ΑΠΘ

Κτίριο Γραμματειών Σχολής Θετικών Επιστημών, Πανεπιστημιούπολη ΑΠΘ

54124 Θεσσαλονίκη

Οι εγκαταστάσεις του Τμήματος στην Καλαμαριά βρίσκονται στην Εθνικής Αντιστάσεως 16. Περισσότερες πληροφορίες στην ιστοσελίδα του Τμήματος:
<https://www.csd.auth.gr/school/access/>

Ερώτηση: Ποιες είναι οι ώρες λειτουργίας της γραμματείας;

Απάντηση: Η Γραμματεία εξυπηρετεί τις ώρες 08:30-15:30, κατά προτίμηση από απόσταση. Σε περίπτωση που απαιτείται η αυτοπρόσωπη προσέλευση στη Γραμματεία, αυτό γίνεται κατόπιν προγραμματισμένου ραντεβού, στις ώρες 11:00 – 12:30 (τηλ. 2310 998420, email: info@csd.auth.gr).

Ερώτηση: Ποιο είναι το ωράριο λειτουργίας της βιβλιοθήκης;

Απάντηση: Το φοιτητικό αναγνωστήριο λειτουργεί 08.00-02.00 (μετά τα μεσάνυχτα) Δευ-Κυρ, τηλέφωνο επικοινωνίας 2310 995354. Μπορείς να βρεις περισσότερες πληροφορίες σχετικά με τη Βιβλιοθήκη εδώ: <https://www.lib.auth.gr/el/επικοινωνία>

Ερώτηση: Ποιες είναι οι ώρες λειτουργίας της λέσχης;

Απάντηση: Η διανομή του συσσιτίου πραγματοποιείται σύμφωνα με το παρακάτω ωράριο: Πρωινό: 8:30-10:00 / Μεσημεριανό: 12:00-16:00 / Δείπνο: 18:00-21:00. Μπορείς να βρεις περισσότερες πληροφορίες σχετικά με τη Λέσχη εδώ: <https://www.auth.gr/weekly-menu/>

Ερώτηση: Πού μπορώ να βρω τον οδηγό σπουδών;

Απάντηση: Μπορείς να βρεις πληροφορίες σχετικά με τον Οδηγό σπουδών εδώ: <https://www.csd.auth.gr/studies/undergraduate/study-guides/>

Ερώτηση: Πού μπορώ να βρω το ακαδημαϊκό ημερολόγιο;

Απάντηση: Μπορείς να βρεις πληροφορίες σχετικά με το Ακαδημαϊκό Ημερολόγιο εδώ: <https://www.csd.auth.gr/news/calendar/>

Ερώτηση: Πού μπορώ να βρω το πρόγραμμα μαθημάτων;

Απάντηση: Μπορείς να βρεις πληροφορίες σχετικά με το Πρόγραμμα μαθημάτων εδώ: <https://www.csd.auth.gr/news/schedules/>

Ερώτηση: Πού μπορώ να βρω το πρόγραμμα εξετάσεων;

Απάντηση: Μπορείς να βρεις πληροφορίες σχετικά με το Πρόγραμμα εξετάσεων εδώ: <https://www.csd.auth.gr/news/schedules/>

Ερώτηση: Πώς μπορώ να κάνω δήλωση μαθημάτων;

Απάντηση: Μπορείς να κάνεις δήλωση μαθημάτων εδώ: <https://students.auth.gr/#/registrations/courses/overview>. Από την ηλεκτρονική γραμματεία [students.auth](https://students.auth.gr/#/registrations/courses/overview) -> Δηλώσεις -> Δήλωση μαθημάτων. Δηλώσεις μαθημάτων γίνονται κάθε εξάμηνο (χειμερινό και εαρινό) τις ημερομηνίες σύμφωνα με τις ανακοινώσεις του Τμήματος. Δεν χρειάζεται να γίνει ξεχωριστή δήλωση για επαναληπτικές εξετάσεις Σεπτεμβρίου.

Ερώτηση: Πού μπορώ να βρω τις ανακοινώσεις του Τμήματος;

Απάντηση: Μπορείς να βρεις ανακοινώσεις του Τμήματος εδώ: <https://www.csd.auth.gr/category/announcements/>

Ερώτηση: Ποιες είναι οι προσφερόμενες ροές μαθημάτων;

Απάντηση: Μπορείς να βρεις πληροφορίες σχετικά με τις Ροές μαθημάτων στον Οδηγό σπουδών εδώ: <https://www.csd.auth.gr/studies/undergraduate/study-guides/>. Οι προσφερόμενες ροές μαθημάτων είναι:

- A. Τεχνητή Νοημοσύνη
- B. Διαχείριση Δεδομένων και Ιστού
- Γ. Τεχνολογίες Λογισμικού
- Δ. Τεχνολογίες Μάθησης
- Ε. Επιστημονική Υπολογιστική
- ΣΤ. Δίκτυα Επικοινωνιών και Ασφάλεια Συστημάτων
- Z. Υπολογιστικά και Επικοινωνιακά Συστήματα
- Η. Ψηφιακά Μέσα – Υπολογιστική Νοημοσύνη
- Θ. Ρομποτική-Αυτόνομα Συστήματα

Για τη λήψη του πτυχίου είναι υποχρεωτική η κατοχύρωση τουλάχιστον μιας (1) ροής μαθημάτων από τις ακόλουθες που προσφέρονται στο Τμήμα.

Ερώτηση: Πού μπορώ να βρω πληροφορίες σχετικά με την πρακτική άσκηση;

Απάντηση: Μπορείς να βρεις πληροφορίες σχετικά με την Πρακτική Άσκηση εδώ: <https://www.csd.auth.gr/studies/undergraduate/praktiki-askisi/>

Ερώτηση: Πού μπορώ να βρω πληροφορίες σχετικά με την εκπόνηση πτυχιακής εργασίας;

Απάντηση: Μπορείς να βρεις πληροφορίες σχετικά με την εκπόνηση Πτυχιακής εργασίας εδώ: <https://www.csd.auth.gr/studies/undergraduate/thesis-guide/>

Ερωτήσεις εκτός πεδίου

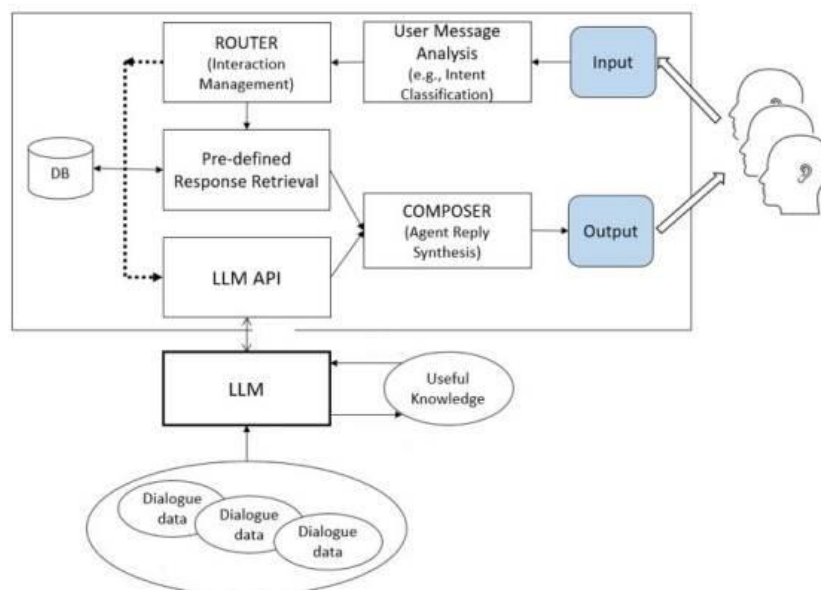
Για τις ερωτήσεις που δεν σχετίζονται με το Τμήμα Πληροφορικής, το Chatbot θα λαμβάνει απαντήσεις από το ChatGPT μέσω API.

3. Rasa framework

4. Υλοποίηση και Αποτελέσματα

4.1 Αρχιτεκτονική της εφαρμογής

Το Chatbot που θα αναπτύξουμε θα έχει παρακάτω αρχιτεκτονική:



Δηλαδή Chatbot παίρνει ως είσοδο ερώτηση του χρήστη , η οποία επεξεργάζεται από το User Message Analysis για να αναγνωριστεί η έννοια και κατηγορία της ερώτησης. Στη συνέχεια, ο Router αποφασίζει αν η ερώτηση πρέπει να απαντηθεί με μια προεπιλεγμένη απάντηση ή αν χρειάζεται η παρέμβαση του LLM. Αν χρειάζεται η παρέμβαση του LLM, τότε γίνεται επικοινωνία μέσω API για να δημιουργήσει την κατάλληλη απάντηση. Τέλος, ο Composer συνδυάζει τις απαντήσεις και τις επιστρέφει στον χρήστη.

Δηλαδή Chatbot δέχεται την ερώτηση του χρήστη αν η ερώτηση είναι σχετίζεται με το Τμήμα Πληροφορικής τότε απαντάει με μια προκαθορισμένη απάντηση, αν όμως η ερώτηση του χρήστη δεν σχετίζεται με το Τμήμα Πληροφορικής, το Chatbot χρησιμοποιεί το API του ChatGPT για να παράγει απάντηση.

Η ενσωμάτωση του ChatGPT API βελτιώνει εμπειρία του χρήστη και μπορεί να προσφέρει πολλές δυνατότητες. Το βασικό πλεονέκτημα αυτής της αρχιτεκτονικής είναι ότι chatbot μπορεί να καλύψει περισσότερο όγκο των ερωτήσεων από πολλές περιοχές και να δώσει ικανοποιητική απάντηση για διάφορα θέματα.

Πιο συγκεκριμένα, το ChatGPT είναι ιδανικό για τις γενικές ερωτήσεις, καθώς έχει εκπαιδευτεί σε ένα μεγάλο όγκο δεδομένων και μπορεί να απαντήσει σε ερωτήσεις από πολλές περιοχές. Ωστόσο, για τις ερωτήσεις που αφορούν ένα συγκεκριμένο τομέα, το chatbot μπορεί να δώσει προκαθορισμένες απαντήσεις, καθώς έχει εξειδικευμένη γνώση και πληροφορία για αυτόν τον τομέα.

Με αυτή την αρχιτεκτονική, μπορούμε να αναπτύξουμε ένα εξειδικευμένο chatbot-βοηθό που μπορεί να δώσει ικανοποιητικές απαντήσεις τόσο για γενικές ερωτήσεις όσο και για ερωτήσεις που αφορούν ένα συγκεκριμένο τομέα. Αυτό σημαίνει ότι μπορούμε να πετύχουμε μια μεγαλύτερη προσωμότητα ενός ανθρώπου-υπάλληλου που έχει εξειδικευμένη γνώση για κάποιο τομέα αλλά μπορεί επίσης να απαντήσει και σε γενικές ερωτήσεις, προσφέροντας μια ολοκληρωμένη και αποτελεσματική εμπειρία χρήστη.

4.2 Περιγραφή του backend κομμάτι

Η ανάπτυξη του Chatbot έγινε στο λειτουργικό σύστημα Windows με την βοήθεια Visual Studio Code IDE.

4.2.1 Εγκατάσταση του Rasa

Πρώτα πρέπει να εγκαταστήσουμε Rasa τοπικά με χρήση της Python 3.10 και virtual environment.

Ελέγχουμε την έκδοση της Python με εντολή “python --version”:

```
PS C:\RasaChatbot> python --version
Python 3.10.11
```

Μετά εγκαταστήσουμε τη δυνατότητα δημιουργίας virtual environments σε Python, με την εντολή “python -m pip install virtualenv”:

```
PS C:\RasaChatbot> python -m pip install virtualenv
Collecting virtualenv
  Downloading virtualenv-20.26.2-py3-none-any.whl (3.9 MB)
    3.9/3.9 MB 698.5 kB/s eta 0:00:00
Requirement already satisfied: platformdirs<5,>=3.9.1 in c:\users\user\appdata\local\programs\python\python310\lib\site-packages (4.2.0)
Requirement already satisfied: distlib<1,>=0.3.7 in c:\users\user\appdata\local\programs\python\python310\lib\site-packages (0.3.8)
Requirement already satisfied: filelock<4,>=3.12.2 in c:\users\user\appdata\local\programs\python\python310\lib\site-packages (3.13.4)
Installing collected packages: virtualenv
Successfully installed virtualenv-20.26.2
```

Για να δημιουργήσουμε ένα virtual environment, τρέχουμε την εντολή “python -m virtualenv venv”:

```
PS C:\RasaChatbot> python -m virtualenv venv
created virtual environment CPython3.10.11.final.0-64 in 6449ms
```

Βλέπουμε ότι δημιουργήθηκε ένα φάκελο venv:

```
▼ RASACHATBOT
> venv
```

Για να χρησιμοποιήσουμε το virtual environment που μόλις φτιάξαμε τρέχουμε την εντολή “venv\Scripts\activate”:

```
PS C:\RasaChatbot> venv\Scripts\activate

(venv) PS C:\RasaChatbot>
```

Και μετά κάνουμε εγκατάσταση Rasa με εντολή “pip rasa install”:

```
(venv) PS C:\RasaChatbot> pip install rasa
Collecting rasa
  Downloading rasa-3.6.20-py3-none-any.whl.metadata (28 kB)
```

Και επαληθεύουμε την εγκατάσταση με εντολή “rasa --version”:

```
(venv) PS C:\RasaChatbot> rasa --version
● Rasa Version      :      3.6.20
  Minimum Compatible Version: 3.5.0
  Rasa SDK Version   :      3.6.2
  Python Version     :      3.10.11
  Operating System    :      Windows-10-10.0.19045-SP0
  Python Path         :      C:\RasaChatbot\venv\Scripts\python.exe
```

4.2.2 Δημιουργία του initial project

Για να φτιάξουμε ένα νέο project χρησιμοποιούμε την εντολή “rasa init”:

```
(venv) PS C:\RasaChatbot> rasa init
```

Ακολουθούμε τις οδηγίες για να δημιουργήσουμε ένα initial project με moodbot (<https://github.com/RasaHQ/rasa/tree/main/examples/moodbot>) , το οποίο είναι πάρα πολύ απλό chatbot που απλά ρωτάει για το πώς αισθάνεται ο χρήστης και αντιδρά ανάλογα με την απάντηση του χρήστη.

Επιλέγουμε φάκελο στο οποίο θα γίνει δημιουργία του project:

```
To get started quickly, an initial project will be created.
If you need some help, check out the documentation at https://rasa.com/docs/rasa.
Now let's start! 🐘

? Please enter a path where the project will be created [default: current directory]
? Directory 'C:\RasaChatbot' is not empty. Continue? Yes
```

Και μετά βλέπουμε ότι δημιουργήθηκε το template project:

```
Created project directory at 'C:\RasaChatbot'.
Finished creating project structure.
```

Και δημιουργήθηκαν απαραίτικοι φάκελοι:

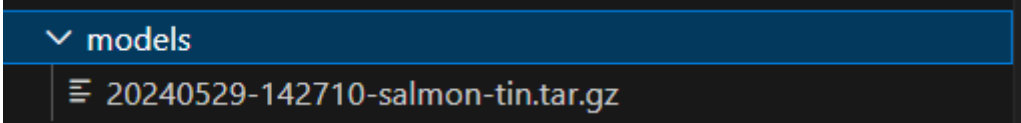
```
✓ RASACHATBOT
  > actions
  > data
  > tests
  > venv
  ! config.yml
  ! credentials.yml
  ! domain.yml
  ! endpoints.yml
```


Μετά μας προτείνουν να κάνουμε training του μοντέλου με εντολή “rasa train”:

```
? Do you want to train an initial model? 🤖 Yes
Training an initial model...
The configuration for policies and pipeline was chosen automatically. It was written into the config file at 'config.yml'.
```

Και βλέπουμε ότι δημιουργήθηκε το μοντέλο και αποθηκεύτηκε στο φάκελο models:

```
Your Rasa model is trained and saved at 'models\20240529-142710-salmon-tin.tar.gz'.
```



A screenshot of a file explorer window. The top bar shows a dropdown menu with 'models' selected. Below it, a list of files is shown, with '20240529-142710-salmon-tin.tar.gz' highlighted.

Και στη συνέχεια μας προτείνουν να αλληλεπιδρούμε με chatbot μέσω γραμμής εντολών τρέχοντας εντολή “rasa shell”:

```
Do you want to speak to the trained assistant on the command line? 🤖 Yes
```

Βλέπουμε ότι ξεκίνησε ο server και μπορούμε να συνομιλήσουμε με το chatbot:

```
2024-05-29 14:37:32 INFO      root - Rasa server is up and running.
Bot loaded. Type a message and press enter (use '/stop' to exit):
Your input -> hi
Hey! How are you?
Your input -> sad
Here is something to cheer you up:
Image: https://i.imgur.com/nGF1K8f.jpg
Did that help you?
Your input -> yes
Great, carry on!
```

4.2.3 Ανάπτυξη του chatbot

Identify questions your chatbot will answer and write corresponding responses. (Chatbot Knowledge Base)	Domain
Anticipate how users will ask questions and provide diverse examples. (User Modeling)	NLU
Connect what users say (intents) with what your chatbot should say in response (action).	DM

Θα ξεκινήσουμε από το domain.yml, ορίζοντας όλα τα intents για κάθε ερώτηση αντίστοιχα

```
intents:
  - greet
  - start
  - goodbye
  - bot_question
  - certificate_question
  - website_question
  - email_question
  - phone_question
  - address_question
  - library_question
  - canteen_question
  - open_hours_question
  - student_guide_question
  - calendar_question
  - schedule_question
  - exam_schedule_question
  - course_registry_question
  - news_question
  - streams_question
  - internship_question
  - thesis_question
  - teacher_info_question
```

Και επεὶτά προσθέτουμε και τις απαντήσεις στα responses:

```
responses:
  utter_greet:
    - text: "Γεια σας ! Πως μπορώ να σας βοηθήσω;"

  utter_goodbye:
    - text: "Καλή συνέχεια. Είμαι πάντα ετοιμος να σας βοηθησω"

  utter_bot:
    - text: "Είμαι SecretaryChatbot που παρέχει πληροφορίες σχετικά με το Τμήμα Πληροφορικής του ΑΠΘ"

  utter_certificate:
    - text: "Μπορείς να πάρεις βεβαίωση σπουδων εδώ: https://students.auth.gr/#/requests/list"
```

Τώρα πρέπει να συμπληρώσουμε το `nlu.yml` που βρίσκεται στο φάκελο `data` με τα παραδείγματα για κάθε intent:

```
nlu:
- intent: greet
  examples: |
    - γεια
    - γεια σας
    - γεια σου
    - καλημερα
    - καλημερα σας
    - καλημερα σου
    - καλησπερα
    - καλησπερα σας
    - καλησπερα σου
    - τι κανεις
    - πως εισαι
    - hi
- intent: start
  examples: |
    - /start
    - start
- intent: goodbye
  examples: |
    - αντιο
    - τα λεμε
    - καληνυχτα
    - καλο βραδυ
    - μπαι
    - bye
```

Καλό είναι να δώσουμε αρκετό αριθμό παραδειγμάτων για κάθε intent για την καλύτερη αναγνώριση από το NLU.

Μετά από αυτό πρέπει να συμπληρώσουμε αρχεία `stories.yml` και `rules.yml`.

!	nlu.yml
!	rules.yml
!	stories.yml

Τα `rules` συνήθως χρησιμοποιούμε για να ορίσουμε σύντομες, καλά καθορισμένες συνομιλίες, όπως χαιρετισμοί, FAQ, απλές φόρμες, που πάντα ακολουθούν την ίδια διαδρομή.

Ενώ τα `stories` αναπαριστούν πιθανά σενάρια συνομιλιών μεταξύ των χρηστών και του chatbot από τα οποία πρέπει να μάθει το chatbot και όχι ως αυστηρές οδηγίες. Έτσι μπορεί να γενικεύσει και να αντιμετωπίσει νέες και απρόβλεπτες καταστάσεις και να έχει μια πιο φυσική και λογική συνομιλία με τους χρήστες.

```
rule: Say hi anytime the user says hi
steps:
- intent: greet
- action: utter_greet

rule: Greet the user in Telegram
steps:
- intent: start
- action: utter_greet

rule: Say goodbye anytime the user says goodbye
steps:
- intent: goodbye
- action: utter_goodbye

rule: Say answer when user say thanks
steps:
- intent: thanks
- action: utter_thanks

rule: Say 'I am a bot' anytime the user ask
steps:
- intent: bot_question
- action: utter_bot
```

Επίσης το Rasa μας δίνει δυνατότητα να δημιουργήσουμε stories με διαδραστική μάθηση με εντολή “rasa interactive”, με την οποία μπορούμε να δημιουργήσουμε τα stories απλά μιλώντας με το chatbot.

Τώρα πρέπει να συμπληρώσουμε το config.yml που περιέχει pipeline για το NLU και policies.

Για να διαχειρίσουμε καλύτερα τα FAQs (συχνές ερωτήσεις) και Chitchat (συνομιλία που δεν σχετίζεται άμεσα με το στόχο, όπως χαιρετισμοί, ευχαριστίες, αποχαιρετισμοί) το Rasa μας προτείνει να χρησιμοποιούμε μια πολιτική διαχείρισης διαλόγου που βασίζεται σε κανόνες (το RulePolicy) και ένα pipeline για να επιστρέψουμε την κατάλληλη απάντηση για μια ερώτηση (το ResponseSelector) (<https://rasa.com/docs/rasa/chitchat-faqs#step-by-step-guide-on-using-response-selector-for-faqs-and-chitchat>):

```
policies:
# # No configuration for policies
# # If you'd like to customize the policies, see the documentation
# # See https://rasa.com/docs/rasa/policies
- name: MemoizationPolicy
- name: RulePolicy

- name: ResponseSelector
  epochs: 100
  retrieval_intent: faq
- name: ResponseSelector
  epochs: 100
  retrieval_intent: chitchat
```

Οπότε πλέον μπορούμε να αντικαθιστούμε όλα τα FAQs για τα φοιτητικά θέματα με ένα rule:

```
- rule: respond to FAQs
  steps:
  - intent: faq
  - action: utter_faq

- rule: respond to chitchat
  steps:
  - intent: chitchat
  - action: utter_chitchat
```

Προσθέτουμε νέα intents “faq” και “chitchat” στο domain.yml και αλλάζουμε ονόματα για τα intents και responses στο nlu.yml και domain.yml:

```
- faq
- chitchat
```

```
- intent: faq/certificate_question
  examples: |
    - βεβαίωση σπουδών
    - πως να παρω βεβαίωση σπουδών
    - μπορείς να δώσεις βεβαίωση σπουδών

- intent: faq/website_question
  examples: |
    - ποια είναι ιστοσελίδα του [τμήματος](facility_type)
    - δώσε οσπ του [τμήματος](facility_type)
    - δώσε σελίδα [τμήματος](facility_type)
    - δώσε site
    - δώσε website

- intent: faq/email_question
  examples: |
    - ποιο είναι email του [τμήματος](facility_type)
    - δώσε διεύθυνση ηλεκτρονικού ταχυδρομείου του [τμήματος](facility_type)
    - δώσε email του [τμήματος](facility_type)
    - email του [τμήματος](facility_type)
    - mail της [γραμματείας](facility_type)
```

```
- faq/certificate_question
- faq/website_question
- faq/email_question
- faq/phone_question
- faq/address_question
- faq/library_question
- faq/canteen_question
- faq/open_hours_question
- faq/student_guide_question
- faq/calendar_question
- faq/schedule_question
- faq/exam_schedule_question
- faq/course_registry_question
- faq/news_question
- faq/streams_question
- faq/internship_question
- faq/thesis_question
- faq/teacher_info_question
- faq
- chitchat
```

Με αυτό τον τρόπο μειώνουμε σημαντικά τον αριθμό των επαναλαμβανόμενων rules για FAQ.

Και τέλος συμπληρώνουμε pipeline και policies στο αρχείο config.yml

(<https://learning.rasa.com/conversational-ai-with-rasa/pipeline/>)

4.2.4 Custom actions

Τώρα θα υλοποιήσουμε ένα custom action που θα μας δίνει την δυνατότητα να επιστρέψει τα στοιχεία του καθηγητή ανάλογα με το επώνυμο που θα δώσει χρήστη.

Στο domain.yml δημιουργούμε entity και slot “teacher”, και επεिता δημιουργούμε μια φόρμα “teacher_info_form”:

```
entities:
  - teacher

slots:
  teacher:
    type: text
    mappings:
      - type: from_entity
        entity: teacher

forms:
  teacher_info_form:
    required_slots:
      - teacher
```

Στο αρχείο rules.yml δημιουργούμε έναν κανόνα για την ενεργοποίηση της φόρμας και ορίζουμε πότε θα εκτελεστεί την φόρμα. Το βήμα active_loop: teacher_info_form δείχνει ότι η φόρμα θα ενεργοποιηθεί μετά την εκτέλεση του teacher_info_form.

```
- rule: Activate Teacher Form
  steps:
    - intent: teacher_info_question
    - action: teacher_info_form
    - active_loop: teacher_info_form
```

Μετά ορίζουμε έναν κανόνα για το τι θα γίνει όταν ο χρήστης θα συμπληρώσει όλα τα slots:

```
- rule: Submit Teacher Form
  condition:
    - active_loop: teacher_info_form
  steps:
    - action: teacher_info_form
    - active_loop: null
    - slot_was_set:
      - requested_slot: null
    - action: action_teacher_info
    - action: action_reset_slot
```

Εδώ βλέπουμε ότι θα εκτελεστούν 2 custom actions: action_teacher_info και action_reset_slot.

Επίσης στο αρχείο domain.yml με ένα response με όνομα της μορφής utter_ask_<slot> μπορούμε να ορίσουμε το μήνυμα με το οποίο chatbot θα ζητάει από τον χρήστη να συμπληρώσει τα απαιτούμενα slots:

```
utter_ask_teacher:  
- text: "Δώσε το επώνυμο του καθηγητή."
```

Ο κώδικας του custom action βρίσκεται στο φάκελο actions στο αρχείο actions.py:



Δημιουργούμε μια κλάση ActionTeacherInfo την οποία θα περιέχει 2 συναρτήσεις:

```
class ActionTeacherInfo(Action):  
    def name(self) -> Text:  
        return "action_teacher_info"  
  
    def run(self, dispatcher: CollectingDispatcher, tracker: Tracker, domain: Dict):  
        # Lookup table for teacher first names and their contact information  
        contact_info = {  
            "Αγγελής": {  
                "email": "leif@csd.auth.gr",  
                "phone": "2310 99-8230"  
            },  
        },
```

Συνάρτηση name πρέπει να επιστρέφει αντίστοιχο όνομα, όπως ορατό δηλώνεται στο αρχείο domain.yml

Και η συνάρτηση run περιέχει κώδικα που αντιστοιχίζει το επώνυμο του καθηγητή με τη λίστα contact_info, αν δεν υπάρχει αντίστοιχο επώνυμο στη λίστα το chatbot θα επιστρέφει σχετικό μήνυμα, αλλιώς θα επιστρέφει τα στοιχεία επικοινωνίας του καθηγητή:

```
# Get the teacher's name provided by the user  
teacher_name = tracker.latest_message['entities'][0]['value'] if tracker.latest_message['entities'] else None  
  
if teacher_name and teacher_name in contact_info:  
    teacher_contact_info = contact_info[teacher_name]  
    response = f"Στοιχεία επικοινωνίας κ. {teacher_name}: \nEmail: {teacher_contact_info.get('email', 'N/A')}\n"  
else:  
    # Use fuzzy matching to find the closest match  
    matched_name, score = process.extractOne(teacher_name, contact_info.keys())  
    if score >= 80: # Adjust the threshold as needed  
        teacher_contact_info = contact_info[matched_name]  
        response = f"Στοιχεία επικοινωνίας κ. {matched_name}: \nEmail: {teacher_contact_info.get('email', 'N/A')}\n"  
    else:  
        response = "Δεν μπορώ να βρω πληροφορίες για αυτόν τον καθηγητή."  
  
dispatcher.utter_message(text=response)  
  
return []
```

4.2.5 Σύνδεση του Chatbot με ChatGPT API

Οι ερωτήσεις που δεν σχετίζονται με το πεδίο γνώσης του chatbot θα απαντηθούν με την βοήθεια του ChatGPT.

Αρχικά το chatbot θα παίρνει ερώτηση από τον χρήστη και θα την αναλήσει, αν πρόκειται για την out-of-scope ερώτηση, τότε θα καλείται custom action μέσω το οποίο chatbot θα επικοινωνεί με την API και στη συνέχεια θα λαμβάνει απάντηση και θα την επιστρέφει στον χρήστη.

Διαχείριση και αναγνώριση out-of-scope ερωτήσεων θα γίνει με την βοήθεια του config.yml, όπου υπάρχει RulePolice. Εδώ βλέπουμε το όριο 0,4, το οποίο σημαίνει ότι αν η βεβαιότητα του NLU για το intent είναι κάτω από το 0,4, τότε θα εκτελεστεί action_default_fallback:

```
- name: RulePolicy
  core_fallback_threshold: 0.4
  core_fallback_action_name: "action_default_fallback"
  enable_fallback_prediction: true
```

Εμείς θα κάνουμε την αλλαγή αυτού του action και θα προσθέσουμε την σύνδεση με API του ChatGPT.

Ο κώδικας για το action θα βρίσκεται στο αρχείο actions.py και θα αποτελεί από μια κλάση ActionDefaultFallback, στην οποία δημιουργούμε μια συνάρτηση load_api_key για να πάρουμε το API-key που βρίσκεται στο αρχείο secret.json και στη συνέχεια κάνουμε σύνδεση στο ChatGPT και στέλνουμε ερώτηση που έδωσε ο χρήστης:

```
class ActionDefaultFallback(Action):
    """Rasa action to parse user text and pulls a corresponding answer
    from ChatGPT."""

    def name(self) -> Text:
        return "action_default_fallback"

    def get_answers_from_chatgpt(self, user_text):

        def load_api_key(secrets_file="secret.json"):
            with open(secrets_file) as f:
                secrets = json.load(f)
                return secrets["OPENAI_API_KEY"]

        # OpenAI API Key from secret.json

        api_key = load_api_key()
        # api_key = api_key
        client = openai.OpenAI(api_key=api_key)

        # Define the prompt
        prompt = f"User: {user_text}\nAI (Δώσε σύντομη απάντηση στα Ελληνικά με μέγιστο 300 tokens): "

        # Call the OpenAI API to generate a completion
        response = client.completions.create(
            model="gpt-3.5-turbo-instruct",
            prompt=prompt,
            max_tokens=300,
            temperature=0.7
        )

        # Extract the generated text from the API response
        generated_text = response.choices[0].text.strip()

        return generated_text
```

Αφού θα πάρουμε απάντηση από το ChatGPT την επιστρέφουμε στον χρήστη στην συνάρτηση run με ένα μήνυμα σημειώνοντας, ότι chatbot έλαβε αυτή την απάντηση από το ChatGPT:

```
def run(self, dispatcher: CollectingDispatcher,  
        tracker: Tracker,  
        domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:  
  
    # Get the latest user text  
    user_text = tracker.latest_message.get('text')  
    response = self.get_answers_from_chatgpt(user_text)  
    # Dispatch the response from OpenAI to the user  
    dispatcher.utter_message('Απάντηση από το ChatGPT: ' + response)  
  
    return []
```

Επίσης προσθέτουμε κάποια παραδείγματα για το τι θεωρούμε out-of-scope ερώτηση στο αρχείο nlu.yml:

```
- intent: out_of_scope_q  
  examples: |  
    - 2 + 2 = ?  
    - που να φάω  
    - ποτε αλλαζουμε ρολοι  
    - τι ωρα ειναι  
    - τι μερα ειναι σημερα  
    - ποια ειναι η πρωτευουσα της ελλαδας
```

Στο αρχείο rules πρέπει να δημιουργήσουμε 2 κανόνες, όπου θα ορίζουμε ότι στην περίπτωση που χρήστης δίνει out-of-scope ερώτηση πρέπει να τρέξουμε action_default_fallback μέσα στο οποίο θα γίνει κλήση του ChatGPT API:

```
- rule: Handle out of scope questions  
  steps:  
    - intent: nlu_fallback  
    - action: action_default_fallback  
  
- rule: Handle out of scope questions 2  
  steps:  
    - intent: out_of_scope_q  
    - action: action_default_fallback
```


4.2.6 Επαλήθευση της λειτουργίας chatbot μέσω γραμμής εντολών

Αφού κάναμε όλα τα παραπάνω βήματα μπορούμε να ελέγξουμε την λειτουργία του chatbot.

Αρχικά με εντολή “rasa train” θα εκπαιδεύουμε ένα μοντέλο χρησιμοποιώντας τα δεδομένα και τις ιστορίες του NLU:

```
PS C:\RasaChatbot> rasa train
```

Το εκπαιδευμένο μοντέλο θα αποθηκευτεί στο φάκελο “models”:

```
Your Rasa model is trained and saved at 'models\20240530-141915-free-cellulose.tar.gz'.
```

Στο αρχείο endpoints.yml πρέπει να ορίσουμε τη διεύθυνση στην οποία θα τρέχει action server:

```
action_endpoint:  
  url: "http://localhost:5055/webhook"
```

Μετά πρέπει πρώτα να ξεκινήσουμε action server με εντολή “rasa run actions”:

```
PS C:\RasaChatbot> rasa run actions
```

Βλέπουμε ότι server ξεκίνησε κανονικά και είναι διαθέσιμος στη θύρα 5055:

```
INFO     rasa_sdk.endpoint - Action endpoint is up and running on http://0.0.0.0:5055
```

Τώρα μπορούμε σε ένα άλλο τερματικό να τρέξουμε εντολή “rasa shell”, που θα φορτώνει εκπαιδευμένο μοντέλο και θα μας επιτρέπει να μιλήσουμε με chatbot στη γραμμή εντολών:

```
2024-05-30 15:13:55 INFO     root - Starting Rasa server on http://0.0.0.0:5005  
2024-05-30 15:13:58 INFO     rasa.core.processor - Loading model models\20240530-141915-free-cellulose.tar.gz...  
2024-05-30 15:15:41 INFO     root - Rasa server is up and running.  
Bot loaded. Type a message and press enter (use '/stop' to exit):  
Your input -> 
```

Παρακάτω θα δούμε ότι τα παραδείγματα που αποδεικνύουν ότι το chatbot δουλεύει σωστά και είναι σε θέση να διαχειρίζεται τις ερωτήσεις πιθανών χρηστών.

Στο πρώτο παράδειγμα ακολουθούμε το σενάριο όπου ο χρήστης χαιρετάει το chatbot, ρωτάει αν είναι όντως chatbot και στη συνέχεια θέτει μια ερώτηση για τον οδηγό σπουδών:

```
Bot loaded. Type a message and press enter (use '/stop' to exit):  
Your input -> γεια σας  
Γεια σας ! Πως μπορώ να σας βοηθήσω;  
Your input -> είσαι μπιτ ?  
Είμαι SecretaryChatbot που παρέχει πληροφορίες σχετικά με το Τμήμα Πληροφορικής του ΑΠΘ  
Your input -> που να βρω οδηγό σπουδών  
Μπορείς να βρεις πληροφορίες σχετικά με οδηγό σπουδών εδώ: https://www.csd.auth.gr/studies/undergraduate/study-guides/  
Your input -> ευχαριστω για την βοήθεια  
Αμα εχετε ερωτησεις είμαι παντα ετοιμος να σας βοηθωω
```

Στο άλλο παράδειγμα θα εξετάζουμε τη λειτουργία custom action για αναζήτησης στοιχείων επικοινωνίας του καθηγητή:

```
Your input -> hi
Γεια σας ! Πως μπορώ να σας βοηθήσω;
Your input -> δωσε στοιχεία επικοινωνιας του καθηγητη
Δώσε το επώνυμο του καθηγητή.
Your input -> Βρακας
Στοιχεία επικοινωνίας κ. Βρακας:
Email: dvrakas@csd.auth.gr
Phone: 2310 99-8885
Your input -> δωσε στοιχεία επικοινωνιας κ. Πολιτη
Στοιχεία επικοινωνίας κ. Πολιτη:
Email: dpolitis@csd.auth.gr
Phone: 2310 99-8406
Your input -> δωσε στοιχεία επικοινωνιας κ. Αβατζη
Δεν μπορώ να βρω πληροφορίες για αυτόν τον καθηγητή.
```

Βλέπουμε ότι η φόρμα λειτουργεί σωστά, καθώς αν δεν δίνουμε αρχικά το επώνυμο του καθηγητή το chatbot θα το ζητήσει από τον χρήστη και μόνο τότε θα αναζητήσει στοιχεία επικοινωνίας του αντίστοιχου καθηγητή, ενώ αν το επώνυμο που έδωσε ο χρήστης δεν αντιστοιχεί σε κάποιο επώνυμο του καθηγητή τότε θα εμφανίζεται σχετικό μήνυμα.

Τέλος θα δούμε την λειτουργία του chatbot στην περίπτωση που χρήστης δίνει ερώτηση που δεν σχετίζεται με το Τμήμα Πληροφορικής, και ο chatbot θα πρέπει να πάρει απάντηση από το ChatGPT:

```
Your input -> hi
Γεια σας ! Πως μπορώ να σας βοηθήσω;
Your input -> τι είναι Rasa
Απάντηση από το ChatGPT: Η Rasa είναι μια ανοιχτού κώδικα πλατφόρμα υποστήριξης συνομιλιών που χρησιμοποιείται για τη δημιουργία chatbots και εφαρμογών τεχνητής νοημοσύνης. Επιτρέπει στους χρήστες να δημιουργήσουν προηγμένους αλγόριθμους συνομιλίας και να ενσωματώσουν φυσική γλώσσα επεξεργασίας σε διάφορες εφαρμογές.
```

Βλέπουμε ότι στο action server έγινε επιτυχής σύνδεση μέσω API, και ο διακομιστής του Open AI επιστρέφει “200 OK” και απάντηση για τον χρήστη εμφανίζεται κανονικά:

```
INFO httpx - HTTP Request: POST https://api.openai.com/v1/completions "HTTP/1.1 200 OK"
```

4.3 Περιγραφή του frontend κομμάτι

Έχουμε τρέξει το chatbot στην γραμμή εντολών, αλλά το Rasa παρέχει πολλές ενσωματωμένες connectors για σύνδεση σε κοινά κανάλια ανταλλαγής μηνυμάτων και φωνής. Μπορούμε επίσης να συνδεθούμε το chatbot με ιστότοπο ή με την εφαρμογή μας με προροθυμισμένα κανάλια REST ή να δημιουργήσουμε το δικό μας προσαρμοσμένο connector (<https://rasa.com/docs/rasa/messaging-and-voice-channels/>).

Εμείς θα ενσωματώσουμε το Chatbot μας στην ιστοσελίδα με την SocketIO κανάλι και πρώτα θα προσθέσουμε credentials στο αρχείο credentials.yml:

```
socketio:
  user_message_evt: user_uttered
  bot_message_evt: bot_uttered
  session_persistence: false
```

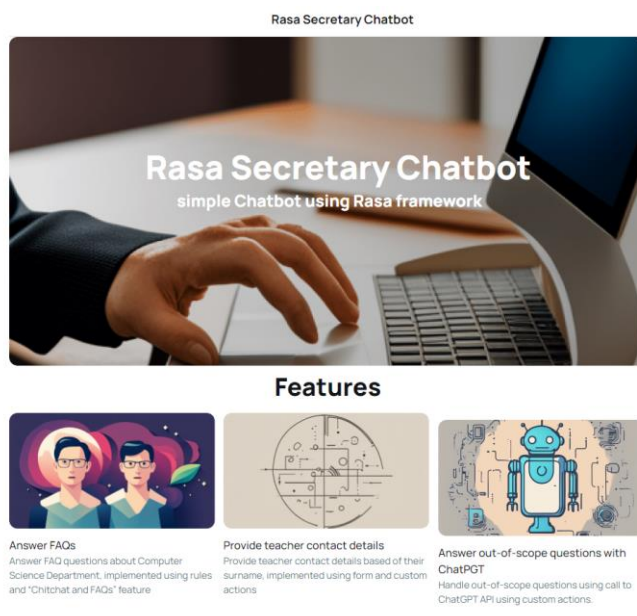
Εδώ οι δύο πρώτες τιμές διαμόρφωσης ορίζουν τα ονόματα συμβάντων που χρησιμοποιούνται από το Rasa κατά την αποστολή ή τη λήψη μηνυμάτων μέσω του socket.io.

Αφού ρυθμίσαμε το κανάλι SocketIO, μπορούμε να χρησιμοποιήσουμε το επίσημο Rasa Chat Widget σε οποιαδήποτε ιστοσελίδα.

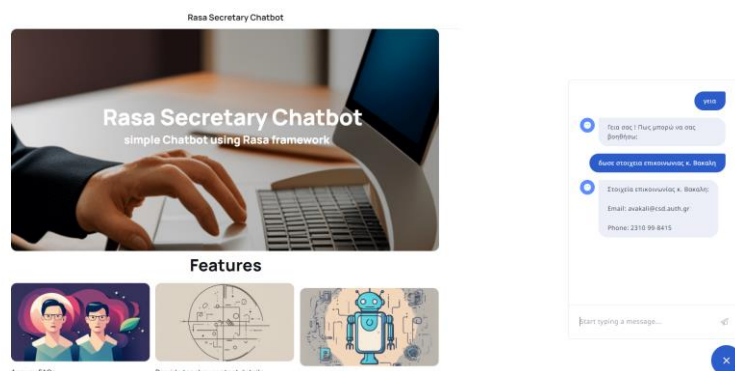
Απλώς βάζουμε αυτό το κώδικα στο HTML της ιστοσελίδας, ορίζοντας τη διεύθυνση για το Rasa server:

```
<div id="rasa-chat-widget" data-websocket-url="http://localhost:5005/"></div>
<script src="https://unpkg.com/@rasahq/rasa-chat" type="application/javascript"></script>
```

Και έτσι το widget εμφανίζεται στην ιστοσελίδα μας:



Μετά ξεκινάμε το action server (με εντολή “`rasa run actions`”) και rasa server (με εντολή “`rasa run -m models --enable-api --cors “*”`”) για να ελέγξουμε ότι το chatbot μας λειτουργεί κανονικά και στην ιστοσελίδα:



Θα δούμε πιο προσεκτικά την εντολή “`rasa run -m models --enable-api --cors “*”`”:

με εντολή «`rasa run`» ξεκινάμε το server με εκπαιδευμένο μοντέλο

«`-m models`» σημαίνει ότι το μοντέλο βρίσκεται στον κατάλογο “models”

«`--enable-api`» σημαίνει ενεργοποίηση της REST API για το μοντέλο

«--cors "*"» σημαίνει ότι επιτρέπονται αιτήσεις από οποιαδήποτε προέλευση (domain, protocol, ή port), χωρίς περιορισμούς CORS.

Τρέχουμε αυτή την εντολή για να αποφεύγουμε πιθανά errors που θα εμφανίζονταν αν τρέχαμε μόνο την εντολή “rasa run”:

```
Access to XMLHttpRequest at 'http://localhost:5005/socket.io/?EIO=4&transport=polling&t=0_BhONf' from origin 'null' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource. index.html:1
```

4.4 Deployment στο Docker Compose

Τώρα θα δούμε πως μπορούμε να κάνουμε το Deployment του chatbot μας με την βοήθεια Docker Compose (<https://rasa.com/docs/rasa/docker/deploying-in-docker-compose/>).

Αρχικά στο φάκελο actions δημιουργούμε το αρχείο Dockerfile:

```
FROM rasa/rasa-sdk:3.6.2

WORKDIR /app
# COPY requirements-actions.txt requirements-actions.txt
COPY . /app
COPY secret.json /app/

# Install extra requirements for actions code
USER root
RUN pip install -r requirements-actions.txt
EXPOSE 5055

USER 1001
```

Και επίσης στο φάκελο actions δημιουργούμε αρχείο requirements-actions.txt με όλα τα imports που ορίζονται στο αρχείο actions.py:

```
> requirements-actions.txt
fuzzywuzzy
python-Levenshtein
requests
openai
pandas
python-dotenv
```

Τώρα δημιουργούμε αντίστοιχο Dockerfile για το rasa server:

```
FROM rasa/rasa-sdk:3.6.2
WORKDIR /app
COPY . /app
COPY secret.json /app/secret.json
USER root
RUN pip install rasa

RUN chmod +x /app/entrypoint-rasa.sh
RUN rasa train --domain /app/domain.yml --data /app/data --out /app/models --debug
ENTRYPOINT ["/app/entrypoint.sh"]
VOLUME /app/models

EXPOSE 5005
```

Επίσης πρέπει να προσθέσουμε αρχείο entrypoint.sh, η οδηγία ENTRYPOINT στο Dockerfile καθορίζει το σενάριο που πρέπει να εκτελεστεί όταν το container ξεκινά. Αυτό επιτρέπει στο container να ρυθμίσει και να εκτελέσει την εφαρμογή Rasa όταν ξεκινά:

```
#!/bin/sh

echo "Activating virtual environment..."
call /app/venv/Scripts/activate.bat

echo "Training Rasa model..."
rasa train

echo "Running Rasa server..."
rasa run --enable-api --cors "*" --debug --endpoints endpoints.yml --log-file out.log --debug --model /app/models/model --port 5005
```

Τέλος δημιουργούμε docker-compose.yml αρχείο, που χρησιμοποιείται για να ορίσουμε και να διαχειριστούμε πολλαπλά Docker containers που εργάζονται μαζί για να λειτουργούν ως μια εφαρμογή:

```
version: '3'
services:
  rasa:
    container_name: "rasa_server"
    user: root
    build:
      context: .
    volumes:
      - "/:/app"
    ports:
      - "5005:5005"

  action_server:
    container_name: "action_server"
    build:
      context: actions
    volumes:
      - ./actions:/app/actions
      - ./data:/app/data
    ports:
      - 5055:5055

  nginx:
    image: "nginx:latest"
    ports:
      - 80:80
    volumes:
      - /:/app
      - ./nginx.conf:/etc/nginx/conf.d/default.conf
```

Επίσης πρέπει να αλλάξουμε το localhost στο όνομα του action_server στο αρχείο endpoints.yml:

```
action_endpoint:  
  url: "http://action_server:5055/webhook"
```

Εκτελούμε εντολή “docker-compose up -d --build” με την οποία το Docker θα δημιουργήσει τις εικόνες για κάθε υπηρεσία που ορίζονται στο αρχείο docker-compose.yml, χρησιμοποιώντας τα αρχεία Dockerfile και θα ξεκινήσει τα containers:

Ανοίγουμε το Docker Desktop, τρέχουμε εντολή “docker-compose up --build”

```
✓ Network rasachatbot_default      Created  
✓ Container rasa_server            Created  
✓ Container action_server          Created  
✓ Container rasachatbot-nginx-1    Created
```

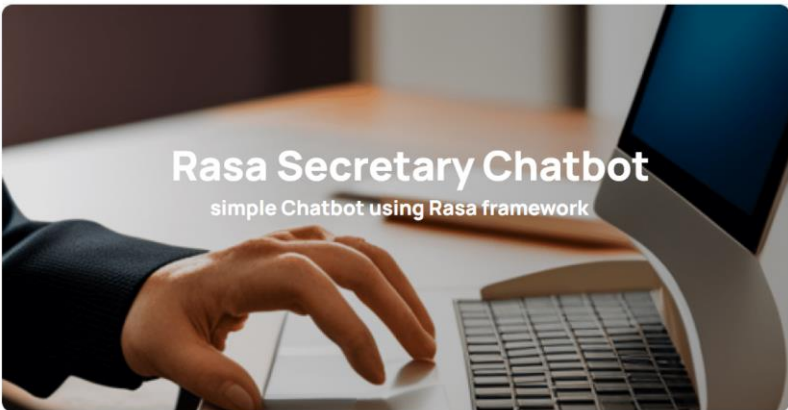
Βλέπουμε ότι δημιουργήθηκαν 3 εικόνες και τα αντιστοίχα containers:

<input type="checkbox"/>	rasachatbot-rasa c11ec94af64e	latest	In use	36 minutes ago	5.41 GB	▶	:	🗑
<input type="checkbox"/>	rasachatbot-action_server a8032f531482	latest	In use	13 minutes ago	443.55 MB	▶	:	🗑
<input type="checkbox"/>	nginx 4f67c83422ec	latest	In use	4 days ago	187.66 MB	▶	:	🗑
<input type="checkbox"/>	📦 rasachatbot		Running (3/3)	0% 9 minutes ago		■	:	🗑
<input type="checkbox"/>	📦 nginx-1 db06acb8f2ad	nginx:latest	Running	80:80	0% 9 minutes ago	■	:	🗑
<input type="checkbox"/>	📦 action_server 0465198072e4	rasachatbot-action_server	Running	5055:5055	0% 9 minutes ago	■	:	🗑
<input type="checkbox"/>	📦 rasa_server 9c4e9f9a7151	rasachatbot-rasa	Running	5005:5005	0% 9 minutes ago	■	:	🗑

Ανοίγουμε localhost και επιβεβαιώνουμε ότι rasa server και action server λειτουργούν σωστά και το chatbot δίνει απαντήσεις σε όλα τα ερωτήματα του χρήστη:

→ ↻ 🔍 localhost




Rasa Secretary Chatbot



Rasa Secretary Chatbot

simple Chatbot using Rasa framework

Features



γεια

Γεια σας! Πως μπορώ να σας βοηθήσω;

3+3

Απάντηση από το ChatGPT: 6

δωσε στοιχ επικ κ. Βακαλη

Στοιχεία επικοινωνίας κ. Βακαλη:
Email: avakali@csd.auth.gr
Phone: 2310 99-8415

Start typing a message...

✕

Το επόμενο βήμα είναι να δημοσιεύσουμε εικόνες μας στο Docker Hub, για αυτό δημιουργούμε το repository:


Create repository

Namespace

kaivanov21

Repository Name *

rasa-chatbot




Short description

Rasa Chatbot repository


A short description to identify your repository. If the repository is public, this description is used to index your content on Docker Hub and in search engines, and is visible to users in search results.

Visibility

Using 0 of 1 private repositories. [Get more](#)

☒ Public 

Appears in Docker Hub search results

☐ Private 

Only visible to you

Cancel

Create

Εμφανίζουμε όλα τα images με εντολή “docker ps -a”:

```
PS C:\RasaChatbot> docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
db06acb8f2ad	nginx:latest	"/docker-entrypoint..."	11 minutes ago	Exited (0) 12 seconds ago		rasachatbot-nginx-1
9c4e9f9a7151	rasachatbot-rasa	"./entrypoint.sh"	11 minutes ago	Exited (137) 2 seconds ago		rasa_server
0465198072e4	rasachatbot-action_server	"./entrypoint.sh sta..."	11 minutes ago	Exited (0) 7 seconds ago		action_server


Βάζουμε τα tags:

```
PS C:\RasaChatbot> docker tag rasachatbot-rasa kaivanov21/rasa-chatbot-rasa:v1
PS C:\RasaChatbot> docker tag nginx kaivanov21/rasa-chatbot-nginx:v1
PS C:\RasaChatbot> docker tag rasachatbot-action_server kaivanov21/rasa-chatbot-action-server:v1
```


Και τώρα μπορούμε να τα ανεβάσουμε στο Docker Hub με εντολή push:



```
docker push kaivanov21/rasa-chatbot-rasa:v1
docker push kaivanov21/rasa-chatbot-nginx:v1
docker push kaivanov21/rasa-chatbot-action_server:v1
```

Μπορούμε να ελέγξουμε το repository και να δούμε διαθέσιμες εικόνες και να επιβεβαιώνουμε ότι εικόνες έχουν προστεθεί με επιτυχία:

kaivanov21/rasa-chatbot 







Updated 2 minutes ago

Rasa Chatbot 

This repository does not have a category   INCOMPLETE

Tags

This repository contains 3 tag(s).

Tag	OS	Type	Pulled	Pushed
 rasa-chatbot-rasa		Image	---	2 minutes ago
 nginx		Image	---	5 minutes ago
 rasa-chatbot-action_s...		Image	---	10 minutes ago

Βλέπουμε ότι εικόνες έχουν δημοσιευθεί και πλέον είναι διαθέσιμα και μπορούν να εγκατασταθούν τοπικά από οποιονδήποτε με την εντολή `pull`.

TAG			
■ rasachatbot-rasa			
Last pushed 7 minutes ago by kaivanov21			
		docker pull kaivanov21/rasa-chatbot:rasachatbot-rasa Copy	
Digest	OS/ARCH	Last pull	Compressed Size ⓘ
0db204f933b6	linux/amd64	---	67.7 MB

TAG			
■ nginx			
Last pushed 10 minutes ago by kaivanov21			
		docker pull kaivanov21/rasa-chatbot:nginx Copy	
Digest	OS/ARCH	Last pull	Compressed Size ⓘ
0db204f933b6	linux/amd64	---	67.7 MB

TAG			
■ rasachatbot-action_server			
Last pushed 15 minutes ago by kaivanov21			
		docker pull kaivanov21/rasa-chatbot:rasachatbot-action_server Copy	
Digest	OS/ARCH	Last pull	Compressed Size ⓘ
e1233a5d1759	linux/amd64	---	163.73 MB

5. Συμπεράσματα

Στο πλαίσιο της παρούσας εργασίας, ολοκληρώθηκε επιτυχώς η ανάπτυξη ενός chatbot με την τεχνολογία Rasa. Το chatbot αναπτύχθηκε σύμφωνα με την αρχιτεκτονική που επιθυμούσαμε. Το chatbot λειτουργεί ικανοποιητικά και καλύπτει ανάγκες των χρηστών, είναι σε θέση να διακρίνει τον τύπο των ερωτήσεων και σε περίπτωση που ερώτηση δεν ανήκει στο πεδίο γνώσης του παίρνει απάντηση από το ChatGPT API.

Επίσης, υλοποιήσαμε την πιλοτική ιστοσελίδα στην οποία ενσωματώσαμε το chatbot μέσω widget, επιτρέποντας στους χρήστες να αλληλεπιδρούν με το chatbot σε ένα φιλικό και εύχρηστο περιβάλλον.

Τελικά, μπορούμε να επιβεβαιώσουμε ότι η Rasa μας δίνει πολλά εργαλεία για την ανάπτυξη του chatbot, όσο και για το deployment. Επίσης, μας δίνει την δυνατότητα σύνδεσης μέσω custom actions με πολλά άλλα εργαλεία μέσω API, επιτρέποντας την επέκταση και την προσαρμογή του chatbot στις ανάγκες των χρηστών.

Το chatbot μπορεί να βελτιωθεί με την προσθήκη νέων λειτουργιών, όπως υποστήριξη δεύτερης γλώσσας ή ενσωμάτωση άλλων συστημάτων για να παρέχει και να παρέχει μια πιο ολοκληρωμένη και εξατομικευμένη εμπειρία χρήστη.