



**UNIVERSITA' POLITECNICA DELLE MARCHE**

**Corso di Laurea Magistrale in Ingegneria Informatica e  
dell'Automazione**

**CORSO DI DATA SCIENCE:  
PROGETTO CHATBOT**

Caprari David  
Cingolani Cristian

Anno Accademico 2021-2022

## Sommario

Introduzione.....	3
Breve introduzione al software Rasa .....	4
Installazione.....	4
Moduli e file di Rasa .....	4
Caso di studio e casi d'uso .....	5
Supporto tecnico .....	5
Supporto finanziario .....	5
Supporto tecnologico.....	6
Semplice <i>chatting</i> .....	6
Realizzazione.....	7
Dominio.....	7
NLU .....	8
Stories.....	8
Rules.....	8
Azioni e interazione col database .....	9
Lettura delle soluzioni tecniche consigliate.....	9
Salvataggio dei ticket.....	10
Note implementative aggiuntive.....	13
Esempio di conversazione .....	14

## Introduzione

Il progetto di cui questo documento è relazione ha come consegne la progettazione e la realizzazione di un chatbot che utilizzi Natural Language Processing e Natural Language Understanding (AI Chatbot) per rispondere a generiche richieste dell'utente attraverso l'utilizzo della suite di strumenti Rasa.

Per i fini di progetto si è scelto di predisporre di un caso di studio molto generico ma che potesse presentare sufficienti sfide progettuali e tecnologiche da giustificare l'utilizzo degli strumenti più smart ed efficaci di Rasa.

Nel dettaglio si è scelto di rispondere alla richiesta di progetto con un chatbot ideato per l'assistenza cliente di una azienda generica. Sono stati generati diversi casi d'uso fittizi relativi alla problematica del fornire una risposta utile al cliente. Si è scelto di suddividere l'assistenza fornita in tre diverse tipologie: finanziaria, tecnica e tecnologica. Ad ognuna di queste, nel caso in cui fosse richiesta, il chatbot dovrebbe rispondere con soluzioni diverse ed adeguate all'esigenza espressa dall'utente utilizzatore, il cliente.

Il backend del chatbot si occupa poi di dialogare con un piccolo database semi-strutturato, anch'esso fittizio, che funge da minima interfaccia di comunicazione esterna al programma. Si è pensato di predisporre un piccolo elenco di prodotti, ai quali sono associate diverse informazioni e a questo si è pensato di associare anche la possibilità di salvare i ticket di interazione col cliente, con l'idea che l'azienda possa servirsene per le sue meccaniche interne di assistenza clienti.

## Breve introduzione al software Rasa

Tralasciando la discussione sui componenti interni alla suite di programmi che ne consentono il funzionamento per esigenze pratiche e di importanza dei contenuti, ci si limiterà a discutere dell'installazione del software e dei moduli che è necessario modificare, in quali modalità, per ottenere il risultato presentato.

Basti sapere che la suite software proposta permette di costruire dei chatbot in grado di comprendere, attraverso l'uso di metodologia ibrida basata sia su predisposizione di regole lessico-sintattiche sia su algoritmi di machine learning, il testo inserito dall'utente. Lo strumento predisposto sarà quindi ovviamente in grado di effettuare tokenizzazione e analisi semantica dei token forniti, riconducendo il tutto a dei campioni, o esempi, di discorso forniti allo strumento precedentemente e con adeguate metodologie pratiche. Nelle sezioni successive descriveremo brevemente il processo di installazione dello strumento, non scontato per quanto abbiamo riscontrato, e daremo una breve introduzione al metodo pratico di generazione dell'agente intelligente descrivendo i file di configurazione dai quali poi lo strumento genera automaticamente il modello operativo.

### Installazione

La suite di software, installata su due macchine separate, è compatibile con i più recenti sistemi operativi.

Per quanto riguarda una delle due macchine, equipaggiata di Windows 11 come OS, si è semplicemente scelto di seguire la guida di installazione fornita dai dottorandi del dipartimento di Ingegneria dell'Informazione dell'Università Politecnica delle Marche<sup>1</sup>.

Per quanto riguarda la seconda delle due macchine, quella dove si è scelto di portare avanti lo sviluppo, l'installazione ha dovuto essere completata in un ambiente GNU/Linux con distribuzione derivata da Arch Linux. L'installazione ha richiesto la predisposizione di un environment Python chiuso con versione dello stesso alla 3.8.13, in quanto Rasa non è compatibile con le distribuzioni più recenti. Per ottenere all'interno di una cartella che funga da environment dedicato un workspace funzionante è stata installata l'estensione di Python denominata Pyenv<sup>2</sup> con il plug-in Pyenv-virtualenv che consente il download pratico ed efficace di diverse distribuzioni con cui poi realizzare diversi environment ognuno di questi associato all'adeguata distribuzione del pacchetto di sviluppo. Terminata la predisposizione degli ambienti, l'installazione ha seguito di nuovo la documentazione ufficiale Rasa<sup>3</sup> con il download del software e l'init di un progetto vuoto.

### Moduli e file di Rasa

Il software Rasa agisce pescando le informazioni di funzionamento da diversi file di configurazione, ognuno contenente settaggi specifici e settoriali.

All'inserimento del comando di inizializzazione, il software sfrutta una procedura guidata per predisporre un primo workspace con un progetto dummy e i file di configurazione necessari. Tralasciando i file `endpoints.yml` e `credentials.yml` che contengono informazioni minime e facilmente compilabili, il processo di realizzazione si basa molto sulla modifica dei file:

- `Domain.yml`: per la generazione del dominio dello scambio in linguaggio naturale tra il bot e l'utente. All'interno di questo file vengono dichiarati i vari intent, possibili stralci di conversazione inseribili dall'utente che il bot dovrà interpretare, le actions con cui il bot dovrà rispondere, quali sono le entità fondamentali all'interno del discorso che si verrà a creare, quali sono i campi interessanti e interattivi che le entità andranno ad occupare (gli slots) e i form da compilare.

---

<sup>1</sup> Disponibile al seguente link: <https://github.com/lucav48/data-science-dataset/blob/master/Rasa/Installazione%20Rasa.md>.

<sup>2</sup> Disponibile al seguente link: <https://github.com/pyenv/pyenv>.

<sup>3</sup> Disponibile al seguente link: <https://rasa.com/docs/rasa/installation>.

- Nlu.yml: un file contenuto nella directory /data che riporta vari esempi testuali precompilati che verranno passati allo strumento di Natural Language Understanding per definire e comprendere gli intent e i sinonimi.
- Stories.yml: file anch'esso nella directory /data che riporta alcuni esempi di possibili path che la conversazione potrebbe prendere, definendo in sostanza le risposte che il chatbot dovrà fornire alla presentazione da parte dell'utente di alcuni intent. Le stories vengono mediate da un sistema di previsione che in base a quanto indicato prevede la risposta successiva del bot.
- Rules.yml: ultimo file rilevante nella directory /data all'interno del quale vanno definite le regole di risposta che il bot deve seguire in ogni caso, bypassando il suo strumento di previsione dello step successivo.
- Actions.py: nel caso in cui il bot debba fornire risposte dinamiche, adattabili al contesto della conversazione, o con la necessità di interagire con strumenti aggiuntivi, quali un dataset o un semplice algoritmo scriptabile, è necessario modificare il file indicato inserendo le procedure richieste.

Compilando adeguatamente questi file, previa progettazione adeguata e testing, si può ottenere un chatbot basato su IA funzionante capace di rispondere alle richieste date dai casi d'uso.

## Caso di studio e casi d'uso

Come già introdotto, il caso di studio preso in esame è relativo alla necessità di predisporre un chatbot capace di rispondere come strumento di indirizzamento all'assistenza utente di una azienda generica.

Si è ipotizzata la necessità di questa azienda di avere a disposizione un servizio utente disponibile 24/7 con adeguata infrastruttura interna che possa aiutare la risoluzione delle problematiche che si possono presentare da parte degli operatori umani.

Si è quindi passati alla determinazione di quattro casi d'uso generici.

### Supporto tecnico

Con il primo caso d'uso si assume che il cliente possa aver comprato un prodotto dell'azienda in esame e abbia necessità di riparazione o di chiarimenti sul funzionamento dello stesso a causa di un possibile guasto. Si è anche ipotizzato che, dato il prodotto nel dettaglio, l'azienda possa già conoscere proattivamente alcune delle problematiche più comuni, e che quindi provi a fornire, prima dell'apertura di un vero e proprio ticket di assistenza, una prima soluzione banale ma frequente, utile a diminuire intelligentemente le richieste di supporto oggettive.

A questo punto, fornite al cliente alcune probabili soluzioni al suo problema, se il cliente ha ancora necessità di supporto o assistenza, è necessario il supporto di un operatore umano. Quindi, si è previsto che si possa generare un ticket di assistenza su cui poi il reparto apposito interno all'azienda possa muoversi fornendo assistenza reale. Per poterlo fare si ha ovviamente la necessità di memorizzare alcune informazioni del cliente, in modo che questo possa essere ricontattato da un operatore.

Tale ticket dovrà ovviamente essere memorizzato su un supporto software esterno al chatbot stesso; ci sarà quindi bisogno di un collegamento logico-pratico tra due strumenti di tipo software.

### Supporto finanziario

Il caso d'uso più semplice rimarca il fatto che il cliente potrebbe aver necessità di rivedere le sue fatture, predisporre il pagamento di una quota, o più in generale, avere a che fare con il reparto acquisti e finanze dell'azienda generica. Data la complessità delle problematiche di questo tipo e nell'intento di differenziare a livello progettuale, si è voluto assumere che la risoluzione di questo tipo di problematiche sia fattibile solo grazie all'apporto di un operatore umano.

A questo punto, al cliente sarà semplicemente richiesto di compilare un ticket con i suoi dati personali in modo da essere opportunamente ricontattato in seguito.

### Supporto tecnologico

Vista l'ampio utilizzo negli ultimi anni di strumenti di questo tipo per l'aiuto a varie componenti aziendali, si è scelto in fase di definizione dei casi d'uso di includerne anche uno che concettualmente possa provare a muoversi esternamente dal campo della semplice assistenza ai clienti.

La richiesta di supporto tecnologico, infatti, interrogherà il cliente in modo che questo fornisca un suo dubbio o una sua curiosità nella fase di acquisto di un prodotto, come se dovesse chiedere informazioni ad un commesso all'interno di un negozio.

A questo punto, il software rimanderà comunque ad un operatore umano la risoluzione di un quesito, tramite la compilazione di un campo testuale apposito all'interno del quale il cliente possa inserire la sua richiesta scritta.

Si è scelto di operare in questa maniera per non appesantire eccessivamente il carico di lavoro relativo al progetto.

### Semplice *chatting*

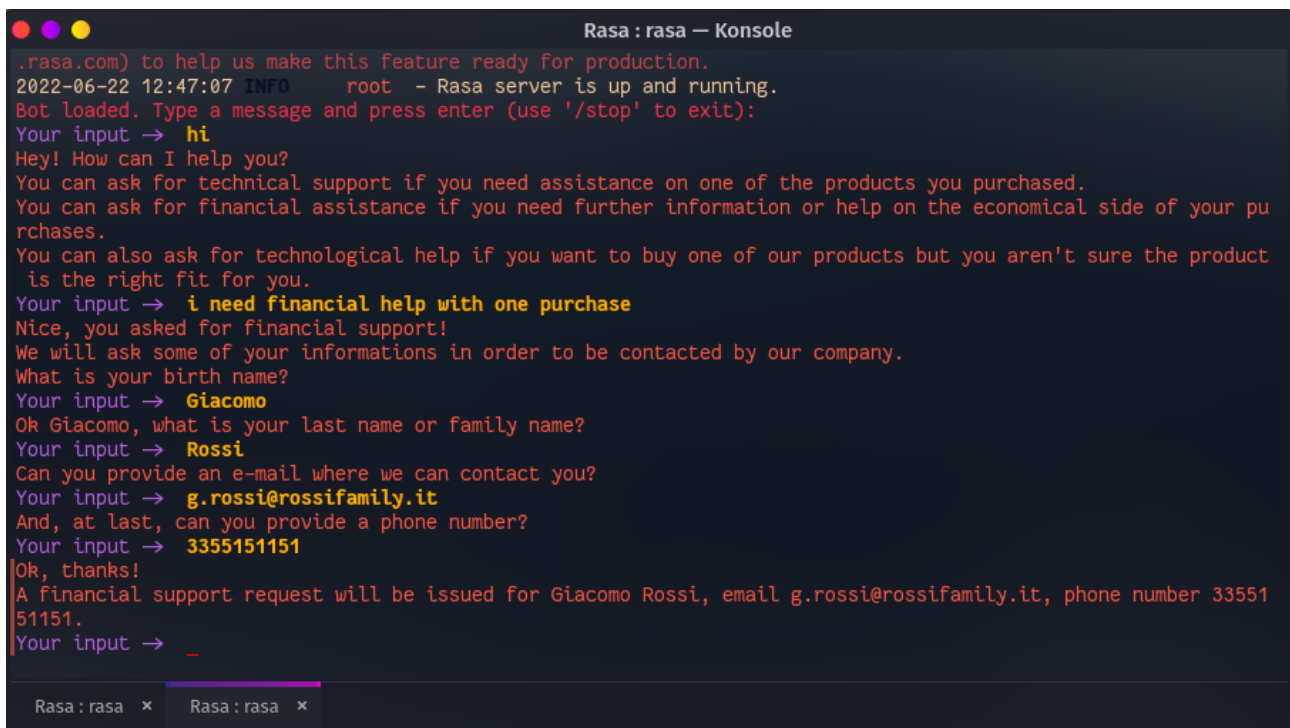
Con caso d'uso di "semplice *chatting*" si vogliono racchiudere tutte quelle attività predefinite fornite già all'inizializzazione del progetto Rasa e qualche minima attività aggiuntiva che il cliente possa essere in grado di svolgere grazie allo strumento.

Si ha infatti la possibilità di inserire il proprio stato d'animo affinché il chatbot possa consolare scherzosamente l'utente o di richiedere l'orario attuale affinché lo strumento risponda.

Ovviamente tale caso d'uso non rientra strettamente nelle richieste di progetto, ma tali componenti che ne portano a termine l'attività sono stati mantenuti, in quanto forniscono comunque informazioni aggiuntive in fase di training.

## Realizzazione

Come già descritto, per l'implementazione dello strumento è avvenuta attraverso la compilazione dei file di configurazione introdotti precedentemente.



```
Rasa : rasa — Konsole
.rasa.com) to help us make this feature ready for production.
2022-06-22 12:47:07 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 can I help you?
You can ask for technical support if you need assistance on one of the products you purchased.
You can ask for financial assistance if you need further information or help on the economical side of your purchases.
You can also ask for technological help if you want to buy one of our products but you aren't sure the product is the right fit for you.
Your input → i need financial help with one purchase
Nice, you asked for financial support!
We will ask some of your informations in order to be contacted by our company.
What is your birth name?
Your input → Giacomo
Ok Giacomo, what is your last name or family name?
Your input → Rossi
Can you provide an e-mail where we can contact you?
Your input → g.rossi@rossifamily.it
And, at last, can you provide a phone number?
Your input → 3355151151
Ok, thanks!
A financial support request will be issued for Giacomo Rossi, email g.rossi@rossifamily.it, phone number 3355151151.
Your input → _
```

Fig. 1: Esempio di conversazione con lo strumento ultimato a riga di comando.

L'implementazione fornita dallo strumento richiede come chiave metodologica principale quella di inserire opportunamente le caratteristiche del dominio trattato, le stories intese come successione di avvenimenti generici, le rules anch'esse generiche e le azioni da svolgere nei file appositi.

A questo proposito riporteremo brevemente quali sono le chicche tecnologiche più interessanti che sono state inserite secondo il gruppo di progetto, analizzando la chiave di progettazione dei diversi file.

## Dominio

All'interno del file che riporta le configurazioni di dominio, oltre ad inserire *intents* ed *actions* sono state inserite diverse entità ritenute interessanti per la risoluzione dei tre principali casi d'uso:

- Phone\_number: riporterà il numero telefonico del cliente per la compilazione dei ticket.
- Support\_type: entità che permetterà all'utente di definire il tipo di supporto di cui ha necessità.
- Name: riporterà il nome del cliente per la compilazione dei ticket.
- Surname: riporterà il cognome del cliente per la compilazione dei ticket.
- E-mail: riporterà l'indirizzo e-mail del cliente per la compilazione dei ticket.
- Product\_id: permetterà al software di ricercare possibili soluzioni ai problemi di un determinato prodotto, nonché di riportarlo nei ticket di assistenza tecnica.
- User\_needs\_technical\_form: sorta di variabile di supporto che permette al software di codificare la scelta di un utente di richiedere comunque assistenza tecnica, indipendentemente dalle soluzioni fornite anticipatamente.

Nonostante sia possibile definire entità semplici per l'andamento della conversazione, la totalità di quelle riportate sono legate ad uno *slot*, strumento di Rasa che funge da contenitore di variabile poi passabile alle azioni che il software dovrà avere implementate.

Tali slot, vengono però caricati attraverso il parsing e l'understanding del contenuto testuale fornito dall'utente, con la possibilità che l'utilizzatore fornisca più informazioni di quante richieste o che le fornisca al momento errato. Per ovviare questo problema, abbiamo, per ogni entità, definito più di una metodologia di caricamento della stessa, indicata dal software come *mapping*.

Un breve esempio:

Supponiamo che l'utilizzatore inizi la conversazione fornendo il suo nome.

“Ciao, mi chiamo Paolo e ho bisogno di assistenza”

Il software, opportunamente progettato sarà in grado di comprendere che 'Paolo' è un nome di una persona, e lo posizionerà quindi nell'entità adeguata, nonostante non ci si trovi nel punto della conversazione in cui il nome è richiesto esplicitamente, come ad esempio la compilazione del form che consentirà di essere ricontattato. Nel caso poi la conversazione finisca nel punto preciso in cui sarebbe stato richiesto il nome dell'utente, essendo questo già stato fornito, il software inserirà automaticamente l'entità all'interno del form, saltandone la richiesta.

Ad estensione del meccanismo appena riportato, nel caso in cui un utente compili più di un form successivamente, seguendo l'idea che un AI Chatbot debba essere in grado di ricordare le interazioni passate, non verranno richiesti i campi di cui il chatbot è già a disposizione.

All'interno del documento relativo ai domini, infine, vengono definiti i form e il comportamento che il software intelligente deve adottare per la loro risoluzione, in particolare quali sono gli intent da scartare e quali sono gli slot da riempire. In questo modo il software andrà a pescarsi da solo domande e risposte che deve fornire all'utente al fine di ottenere i campi.

Infine, il file contiene gli utter di risposta.

## NLU

Il file viene utilizzato per fornire alcuni esempi al software che si occuperà di addestrare il modulo di Natural Language Understanding. Seguendo le indicazioni di buono utilizzo fornite dalla documentazione software, tale file è stato compilato attraverso l'utilizzo della routine interattiva di addestramento, che consente di fornire e generare stories, rules ed esempi per l'NLU automaticamente.

## Stories

Allo stesso modo del precedente, in maniera fortemente consigliata dalla documentazione software nel caso dell'utilizzo di forms, il file è stato compilato inizialmente utilizzando la routine interattiva di addestramento. Con l'utilizzo di tale routine più e più volte durante il processo di sviluppo del software si è permesso allo strumento di memorizzare un ottimo numero di esempi di conversazione.

Tale processo, a giudizio del team di progetto, è molto utile per permettere allo strumento di evolvere durante il testing, ottimizzando lo sviluppo.

Nonostante questo, però, è stato necessario effettuare pulizia delle entry inserite automaticamente, poiché alcune entità o intent non vengono riconosciuti correttamente, nonostante lo strumento finale non perda comunque il senso del discorso.

## Rules

Con il proposito di ottenere azioni specifiche *esterne* da parte del software, è stato possibile utilizzare la configurazione delle regole in modo da inserirvi la chiamata all'azione richiesta.

Ne è un esempio il processo di scrittura su database semi-strutturato (che argomenteremo di seguito). La chiamata alla funzione che effettuerà la scrittura esterna va chiamata solo e soltanto dopo aver ottenuto le informazioni da scrivere. Utilizzando questo file di configurazione nelle modalità adeguate è stato possibile realizzare quanto richiesto.



## Azioni e interazione col database

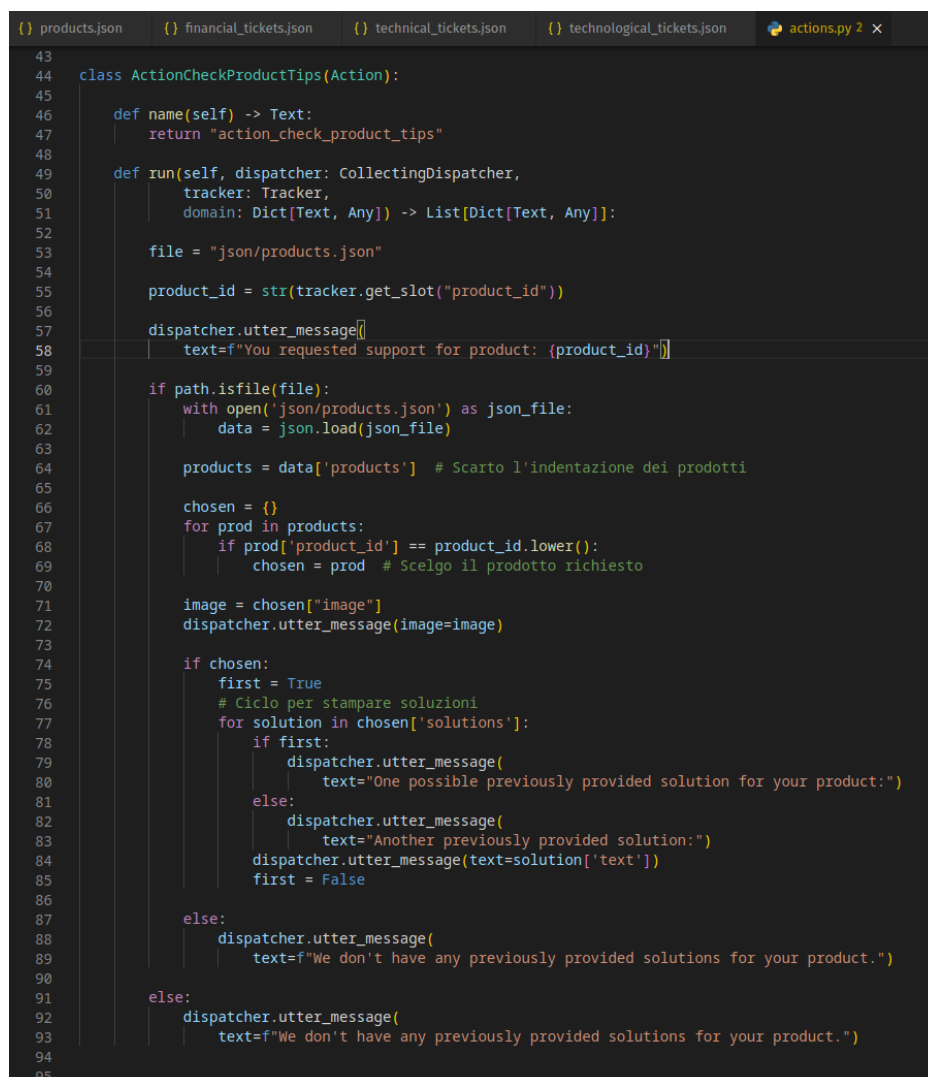
Al fine di ottenere le interazioni descritte precedentemente, si è scelto di scrivere alcuni file .JSON che potessero fare al caso nostro nella necessità di estrazione di alcune info sui prodotti o nella necessità di salvare il ticket generato dall'interazione con il cliente.

Si è scelto un semplice salvataggio su file per evitare la maggior complicazione di dover provvedersi di un DBMS per la gestione delle query ad un database convenzionale, ma non si esclude che tale implementazione possa essere possibile, anzi, la si consiglia per casi di studio più complessi del nostro.

## Lettura delle soluzioni tecniche consigliate

Di seguito si riporta un'immagine con il codice implementato per l'accesso al prodotto problematico fornito dal cliente. Nel caso in cui questo sia presente nel database, vengono fornite alcune soluzioni precedentemente compilate nel database dei prodotti, il file *products.json*, come già anticipato.

Per dimostrare le capacità dello strumento, il database dei prodotti contiene anche un semplice link ad alcune immagini, che in fase di conversazione verranno opportunamente mostrate al cliente.



```
43 class ActionCheckProductTips(Action):
44
45     def name(self) -> Text:
46         return "action_check_product_tips"
47
48     def run(self, dispatcher: CollectingDispatcher,
49             tracker: Tracker,
50             domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
51
52         file = "json/products.json"
53
54         product_id = str(tracker.get_slot("product_id"))
55
56         dispatcher.utter_message(
57             text=f"You requested support for product: {product_id}")
58
59         if path.isfile(file):
60             with open('json/products.json') as json_file:
61                 data = json.load(json_file)
62
63                 products = data['products'] # Scarto l'indentazione dei prodotti
64
65                 chosen = {}
66                 for prod in products:
67                     if prod['product_id'] == product_id.lower():
68                         chosen = prod # Scelgo il prodotto richiesto
69
70                 image = chosen["image"]
71                 dispatcher.utter_message(image=image)
72
73                 if chosen:
74                     first = True
75                     # Ciclo per stampare soluzioni
76                     for solution in chosen['solutions']:
77                         if first:
78                             dispatcher.utter_message(
79                                 text="One possible previously provided solution for your product:")
80                         else:
81                             dispatcher.utter_message(
82                                 text="Another previously provided solution:")
83                             dispatcher.utter_message(text=solution['text'])
84                             first = False
85                 else:
86                     dispatcher.utter_message(
87                         text=f"We don't have any previously provided solutions for your product.")
88
89         else:
90             dispatcher.utter_message(
91                 text=f"We don't have any previously provided solutions for your product.")
92
93
94
95
```

Fig. 2: Implementazione della funzione di lettura dal database dei prodotti.

Di seguito viene riportato il database dei prodotti, che per semplicità contiene solo due entry.

```
products.json x financial_tickets.json technical_tickets.json technological_tickets.json
1  {
2    "products": [
3      {
4        "product_id": "id001",
5        "solutions": [
6          {
7            "code": "11",
8            "text": "If the user is getting error 'e11' try rebooting the device."
9          },
10         {
11           "code": "12",
12           "text": "If the user is getting error 'e12' try to give the device a Fonzi's hit."
13         },
14         {
15           "code": "13",
16           "text": "If the user is getting error 'e13' try to jump on the device."
17         },
18         {
19           "code": "14",
20           "text": "If the user is getting error 'e13' try to throw the device out of the window."
21         }
22       ],
23       "image": "https://www.netgear.com/cid/fit/1024x633/to/jpg/https://www.netgear.com/media/R7000_productcarousel_3_tcm148-96545.png"
24     },
25     {
26       "product_id": "id002",
27       "solutions": [
28         {
29           "code": "21",
30           "text": "If the user is getting error 'e21' try rebooting the device."
31         },
32         {
33           "code": "22",
34           "text": "If the user is getting error 'e22' try to give the device a Fonzi's hit."
35         },
36         {
37           "code": "23",
38           "text": "If the user is getting error 'e23' try to jump on the device."
39         },
40         {
41           "code": "24",
42           "text": "If the user is getting error 'e23' try to throw the device out of the window."
43         }
44       ],
45       "image": "https://images.eprice.it/nobrand/0/Lightbox/542/301153542/1012902971.jpg"
46     }
47   ]
48 }
```

Fig. 3: Schermata del database dei prodotti.

Ciò è stato mostrato per dimostrare che lo strumento scelto rende molto semplice l'interazione con strumenti esterni e che nel caso di ampliamento dei casi d'uso, le possibilità di interazione sono ragguardevoli.

### Salvataggio dei ticket

Come precedentemente descritto, al termine della compilazione del form, quando questo sarà completato e si uscirà dal loop di completamento, affinché l'assistenza ipotizzata dal caso di studio possa essere fornita in maniera ottimale, è necessario che il programma effettui il salvataggio dei ticket all'interno di un file (o un DBMS) esterno.

Il codice che viene allegato effettua proprio quanto appena descritto, salvando, per ogni tipo di ticket, i dati all'interno di un file nella directory di progetto.

```

94
95
96 class ActionSaveTechnicalTicket(Action):
97
98     def name(self) -> Text:
99         return "action_save_technical_ticket"
100
101     def run(self, dispatcher: CollectingDispatcher,
102             tracker: Tracker,
103             domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:
104
105         file = 'json/technical_tickets.json'
106
107         name = tracker.get_slot("name")
108         surname = tracker.get_slot("surname")
109         product_id = tracker.get_slot("product_id")
110         email = tracker.get_slot("email")
111         phone_number = tracker.get_slot("phone_number")
112
113         if path.isfile(file):
114
115             with open(file) as fp:
116                 tickets = json.load(fp)
117                 ticket_id = tickets[-1]['ticket_id']+1
118                 data_ticket = {'ticket_id': ticket_id, 'name': name, 'surname': surname,
119                               'product_id': product_id, 'email': email, 'phone_number': phone_number}
120                 data = tickets + [data_ticket]
121
122         else:
123             data = [{'ticket_id': 0, 'name': name, 'surname': surname,
124                     'product_id': product_id, 'email': email, 'phone_number': phone_number}]
125
126         with open(file, 'w') as outfile:
127             json.dump(data, outfile)
128

```

Fig. 4: Codice per il salvataggio dei ticket tecnici.

È facile notare come al termine della compilazione delle entità, e quindi degli slot, sia semplice caricarne i dati all'interno di uno script. Il salvataggio viene poi eseguito con le opportune query di scrittura su file.

Nell'immagine successiva viene mostrato il database dei ticket tecnici compilati grazie all'utilizzo del software.

```
{ } products.json    { } financial_tickets.json    { } technical_tickets.json x    { } technological_tickets.json

1  [{
2    "ticket_id": 0,
3    "name": "David",
4    "surname": "Caprari",
5    "product_id": "id001",
6    "email": "s1097622@studenti.univpm.it",
7    "phone_number": "3333333333"
8  }, {
9    "ticket_id": 1,
10   "name": "David",
11   "surname": "Caprari",
12   "product_id": "id002",
13   "email": "s1097622@studenti.univpm.it",
14   "phone_number": "3333333333"
15 }, {
16   "ticket_id": 2,
17   "name": "David",
18   "surname": "Rossi",
19   "product_id": "id001",
20   "email": "d.rossi@studenti.it",
21   "phone_number": "3433443343"
22 }, {
23   "ticket_id": 3,
24   "name": "David",
25   "surname": "Prova",
26   "product_id": "id001",
27   "email": "Prova@prova.it",
28   "phone_number": "3"
29 }, {
30   "ticket_id": 4,
31   "name": "David",
32   "surname": "Caprari",
33   "product_id": "id001",
34   "email": "s1097622@studenti.univpm.it",
35   "phone_number": "3333535535"
36 }, {
37   "ticket_id": 5,
38   "name": "Mattia",
39   "surname": "Genovese",
40   "product_id": "Id001",
41   "email": "m.gen@provaprova.it",
42   "phone_number": "1234567890"
43 }, {
44   "ticket_id": 6,
45   "name": "David",
46   "surname": "Caprari",
47   "product_id": "id001",
48   "email": "s1097622@studenti.univpm.it",
49   "phone_number": "3333535535"
50 }]
```

Fig. 5: Database compilato automaticamente dei ticket tecnici.

### Note implementative aggiuntive

Data la semplicità del caso di studio proposto, ai fini della realizzazione del progetto si è scelto di non prevedere una sanificazione dell'input o di implementare controlli eccessivi sull'andamento della conversazione. Questo per non appesantire eccessivamente il carico di progetto. Per questo motivo si tiene a specificare che il software presentato è sufficientemente lontano da un utilizzo funzionale ottimale. Viene infatti dato per appurato il fatto che software di questo tipo richiedono molte iterazioni produttive ed uno sviluppo intensivo.

Affinché sia comunque testabile in maniera ottimale, con il fine aggiuntivo di mostrare le capacità fornite dalla suite Rasa, si è comunque scelto di utilizzare il software Ngrok per aprire pubblicamente una porta della macchina con cui si è sviluppato il chatbot per rendere momentaneamente disponibile il bot sulla piattaforma di messaggistica Telegram.

## Esempio di conversazione

Verrà riportato un breve esempio di conversazione ottenuta dialogando con il chatbot, con l'intento di raccontare nel dettaglio cosa avviene a livello di esecuzione.

Partendo dall'istante zero, il software è stato avviato su una macchina connessa al servizio di Ngrok. All'interno del file `credentials.yml` sono codificati e opportunamente inseriti gli estremi di configurazione del bot di Telegram. Quindi, a questo punto, l'utente cerca il bot della nostra azienda fittizia su nei canali del software di messaggistica e contatta lo strumento.

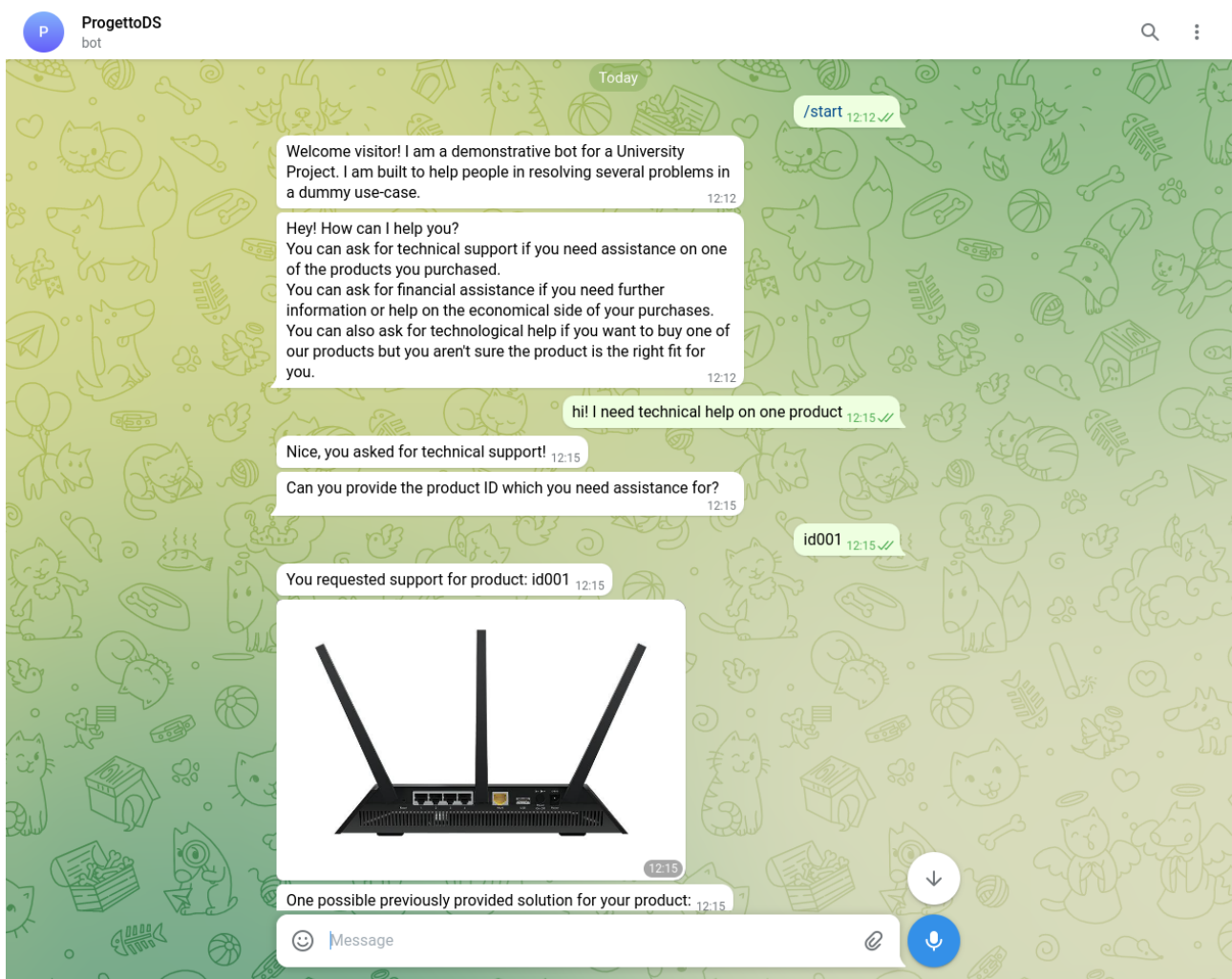


Fig. 6: Prima parte della conversazione di esempio.

All'avvio della conversazione con l'input `/start` tipico di Telegram, il chatbot riconoscerà l'intent `start` adeguato e avvierà una rule in cui viene indicato che all'intent riconosciuto `start` va risposto con l'utter `utter_start` seguito dall'utter `utter_greet`. A questo punto, l'utente indicherà, su consiglio del bot, di richiedere supporto tecnico. Si avvierà quindi la prima parte della routine del supporto tecnico che, se non è ancora stato indicato, chiederà all'utente il prodotto per il quale ha bisogno di assistenza.

Effettuata la lettura dell'identificativo del prodotto, il chatbot avvierà l'action di lettura dal database alla ricerca di eventuali soluzioni precedentemente fornite.

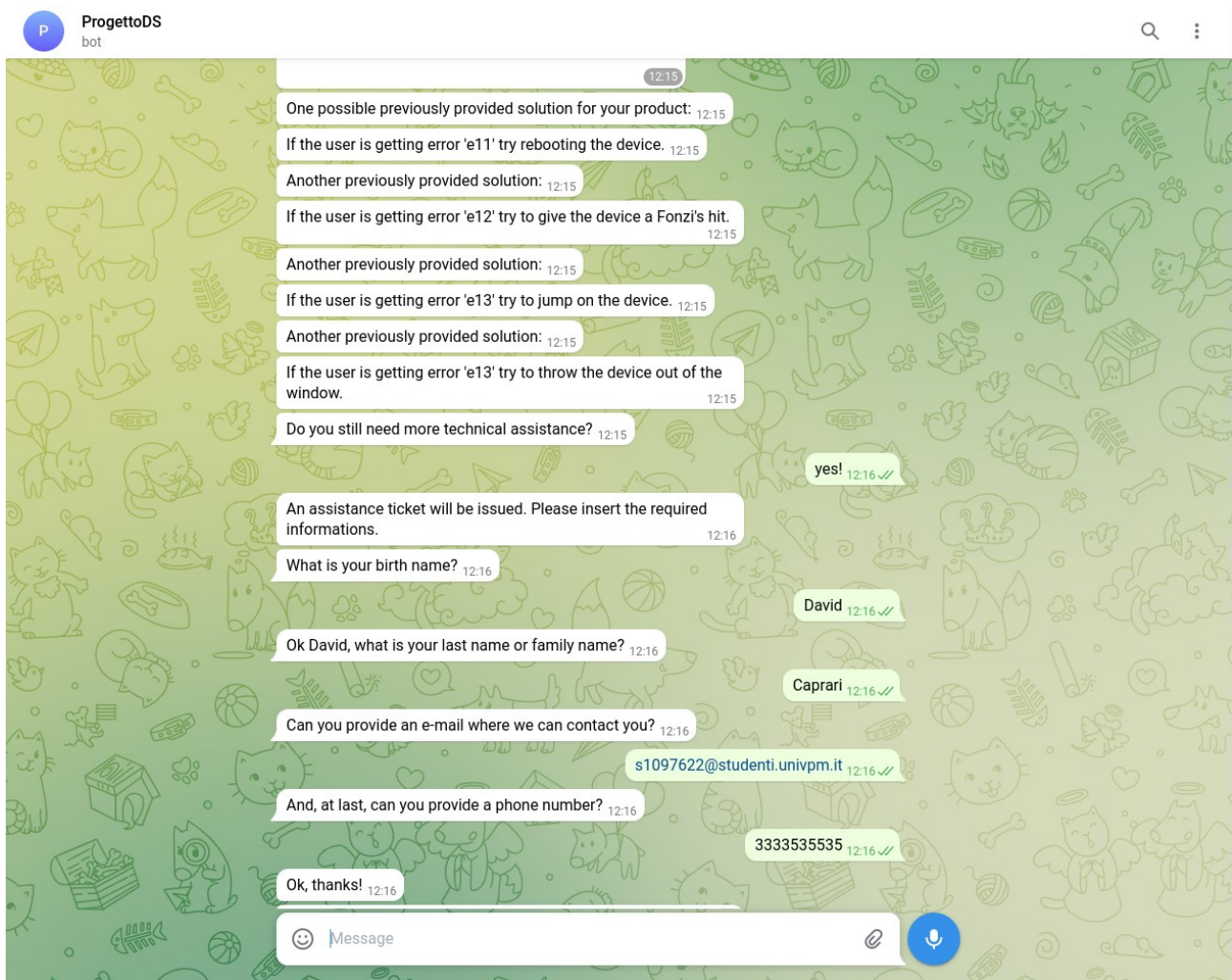


Fig. 7: Seconda parte della conversazione di esempio.

L'esecuzione della routine in Python consentirà quindi al bot di ricercare le soluzioni previste per il relativo prodotto all'interno del database dei prodotti. Nel caso in cui, come nell'esempio, l'utente abbia ancora bisogno di assistenza, il chatbot entrerà nel loop del form tecnico, iniziando a richiedere i campi indicati nella definizione del form stesso.

Partirà quindi con la richiesta dei dati dell'utente, con la possibilità di rendere più interattiva ed attrattiva la conversazione effettuando l'accesso ai dati appena inseriti. Al termine dell'inserimento, compilerà il form e, grazie all'action di salvataggio, memorizzerà i dati sul supporto software previsto, al fine di compilare il ticket relativo. Risponderà quindi con i dati raccolti, a mo' di conferma per l'utente dell'inserimento dei suoi dati.



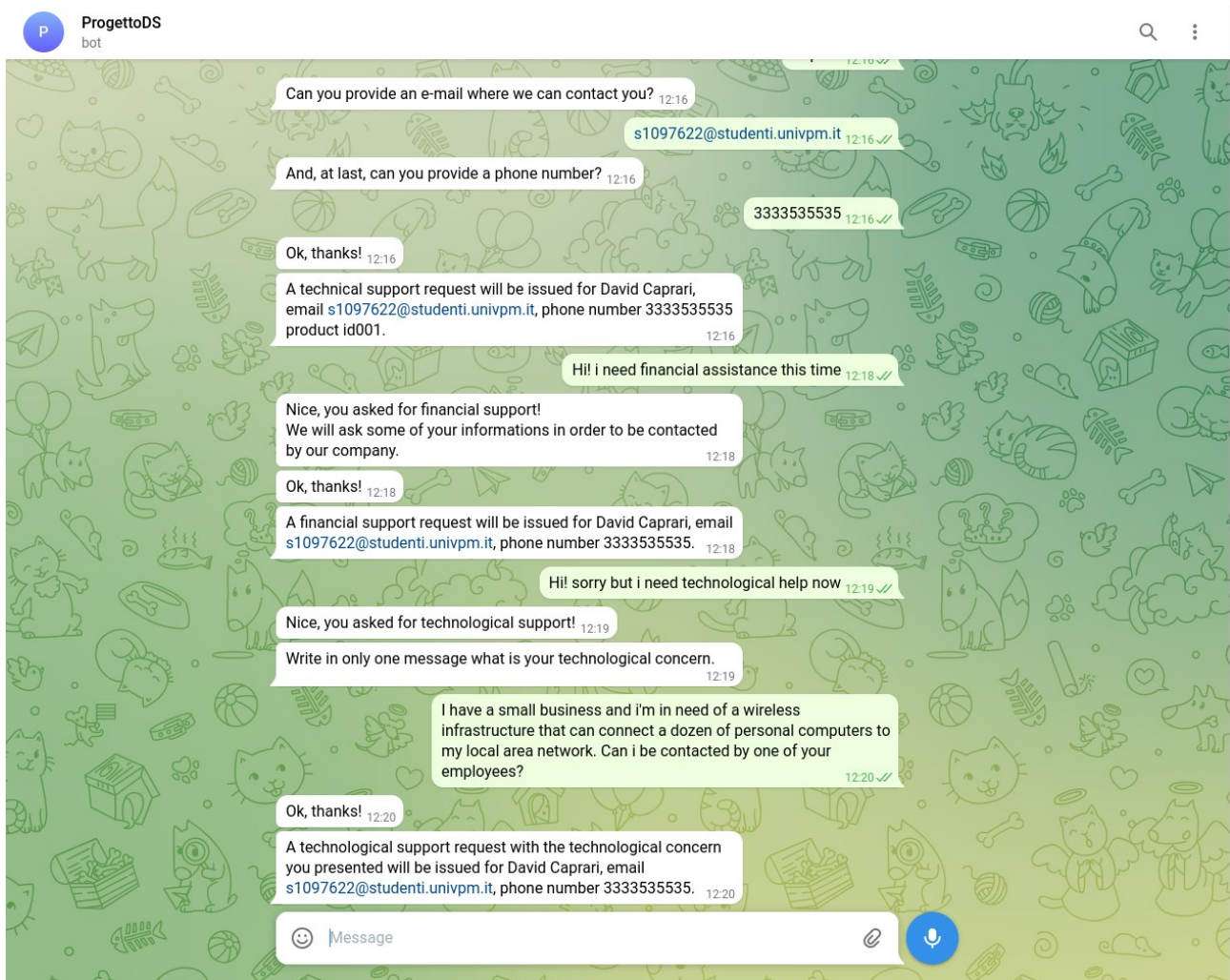


Fig. 8: Terza parte della conversazione di esempio.

Nel caso in cui l'utente voglia presentare nuove problematiche, come già indicato, i dati di contatto, visto che sono già stati inseriti, verranno mantenuti in memoria. Nel momento in cui infatti il cliente indica di aver necessità di supporto finanziario, il chatbot avvia il loop relativo al form finanziario. Ma visto che quest'ultimo richiede soltanto un sottoinsieme dei dati precedentemente inseriti, il software compilerà il form con i dati che ha già in possesso rispondendo immediatamente con "Ok, thanks!" a significare l'effettivo completamento del form.

Lo stesso avviene nel caso di richiesta di assistenza tecnologica, ma, visto che questa volta il loop di inserimento richiede più informazioni di quante già inserite, quelle mancanti verranno esplicitamente richieste e sarà possibile compilare il form adeguatamente.



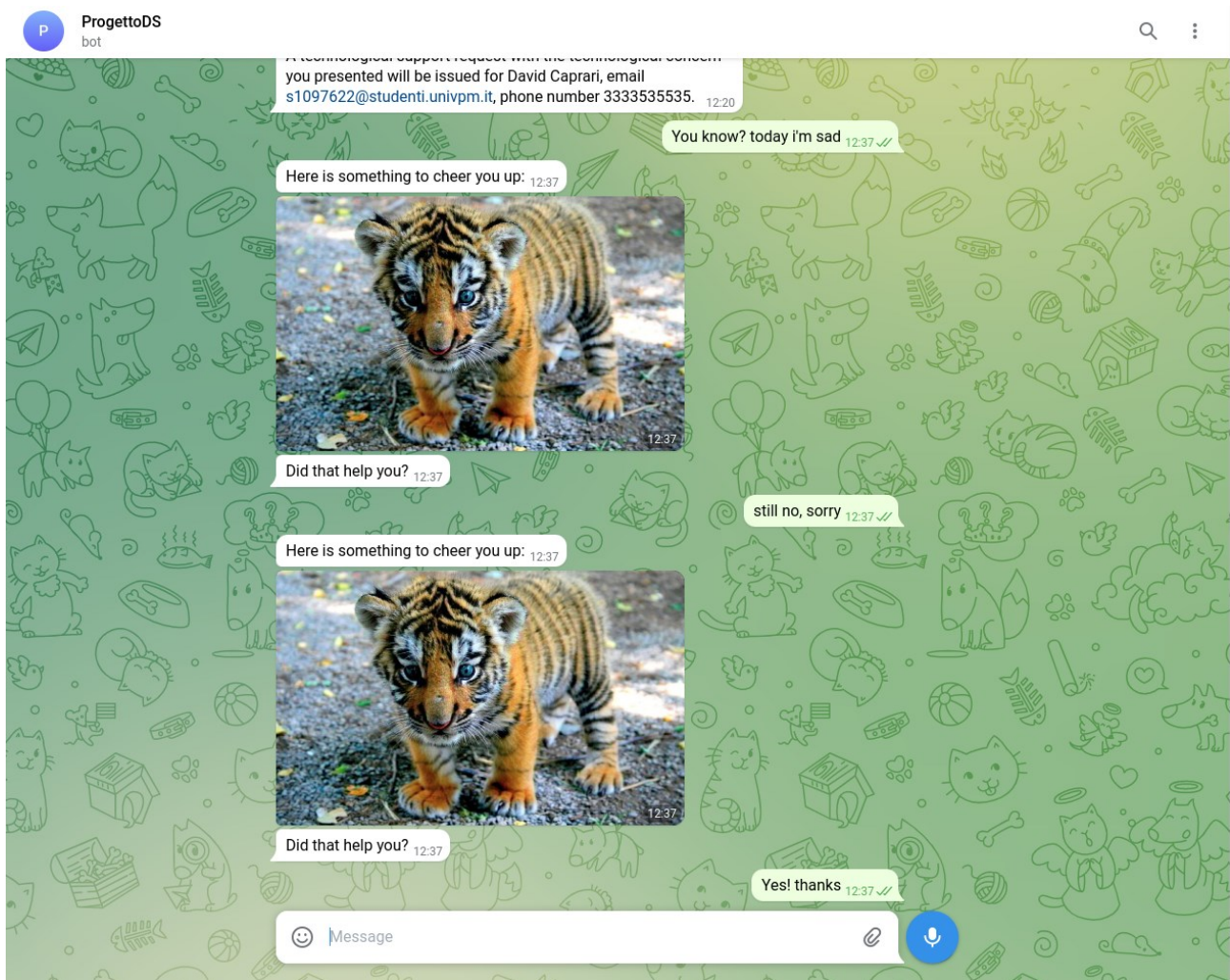


Fig. 9: Ultima parte della conversazione di esempio.

Nell'ultima sezione, mostriamo come, in modalità completamente triviale, sia possibile compilare l'ultimo caso d'uso previsto.