Università Politecnica delle Marche

Facoltà di Ingegneria

Dipartimento di Ingegneria dell'Informazione

Implementazione di un Chatbot per l'orientamento universitario dell'Univpm tramite framework Rasa

Docenti:
Domenico Ursino
Michele Marchetti

Studenti:
Lorenzo Fratini
Federico Miscia
Andrea Pinciaroli

Anno Accademico 2021/2022

Indice

Capitolo 1

Introduzione

La seguente relazione tratterà gli aspetti salienti della progettazione e sviluppo di un chatbot per l'orientamento all'interno dell'Università Politecnica delle Marche. Più precisamente, esso permetterà di ottenere informazioni generiche relative alle tasse universitarie e alle relative scadenze, alle facoltà presenti presso UNIVPM e ai corsi, con il relativo piano studi. Inoltre, permetterà allo studente di orientarsi anche all'interno del plesso universitario fornendo indicazioni sulla posizione della aule di interesse.

1.1 Chatbot & Rasa

Un **chatbot** è un sistema di intelligenza artificiale che permette agli esseri umani di interagire con la tecnologia utilizzando una varietà di metodi di input come voce, testo, gesture e touch, il tutto 24 ore al giorno, 7 giorni alla settimana e 365 giorni all'anno. Oltre alla disponibilità, altri vantaggi nell'utilizzo di un chatbot sono la riduzione dei costi dell'intervento umano, l'alta scalabilità e la compatibilità con molte piattaforme di messaggistica istantanea.

Nel progetto in questione, lo strumento principale utilizzato è stato **Rasa**, un framework dedicato alla costruzione di chatbot conversazionali, il quale sfrutta modelli di machine learning e il Natural Language Processing (NLP) per comprendere l'intento dell'utente e predire la risposta corretta.
La scelta di Rasa è dovuta a diverse ragioni, tra cui il fatto che è disponibile gratuitamente ed open-source, è semplice da usare, è ampiamente flessibile e personalizzabile, ha una grande community di supporto ed infine, ma non per importanza, è on-premises, cioè si può fare il deploy sul proprio server mantenendo tutti i dati in locale il che, di fatto, risolve questioni legate alla privacy.

L'architettura Rasa è costituita da due componenti fondamentali, che sono:

- **RASA NLU**: si occupa di capire e classificare la volontà dell'utente (chiamato anche *intente*), prendendo come input del testo "libero" scritto da quest'ultimo e restituendo dati strutturati

- **RASA CORE**: dopo aver classificato il messaggio dell'utente, elabora quella che sarà la risposta sulla base di ciò che è stato recepito al momento presente, ma anche in passato.

Per quel che riguarda la fase di comprensione e classificazione della richiesta dell'utente, Rasa utilizza un approccio di apprendimento supervisionato. Da qui la necessità di fornire dati di training, in particolare i cosiddetti ***intent***, cioè quelle che sono le possibili motivazioni che portano un utente a rivolgersi al chatbot. Ad ogni frase scritta dall'utente, infatti, corrisponde un'intenzione e l'obiettivo del chatbot, in prima istanza, sarà proprio quello di individuare e classificare correttamente tale aspetto.
Addestrare il chatbot sugli *intent* significa fornire un numero opportuno di esempi per ciascuna tipologia di richiesta che può provenire dall'utente. Di fatto, è necessario immedesimarsi nell'utente e pensare cosa egli potrebbe dire per esprimere un determinato concetto. Per salutare, ad esempio, un utente potrebbe scrivere "Ciao","Salve","Buongiorno" etc... dunque, nel codice, si avrà un *intent* "Saluti" con le principali e più probabili forme di saluto annesse.
Una volta aver incluso tutti i possibili intent relativi al dominio di interesse, il chatbot sarà in grado di classificare un input ricevuto associandogli l'intent che più si addice, sulla base di un punteggio.
Oltre agli *intent*, comunque, è possibile arricchire il training del chatbot attraverso ulteriori elementi, i quali verranno illustrati puntualmente nel corso della trattazione.
Ad ogni modo, una volta che che la componente RASA NLU rileva l'*intent* relativo ad una frase di input dell'utente, entra in gioco RASA Core che crea un modello di machine learning per apprendere dagli esempi forniti e predire la risposta più adatta da restituire all'utente, scegliendola tra le ***utterences***. In alternativa, se la risposta prevede un'elaborazione più articolata, come ad esempio l'invocazione di una API o una query su database, può essere necessario far uso delle ***Actions***.
Tutti i file da utilizzare, fatta eccezione per le azioni, hanno estensione *.yml* e nel paragrafo successivo verrà descritto, più nel dettaglio, come essi sono organizzati in un progetto RASA.

1.2 Design chatbot UNIVPM

Il chatbot di questo progetto si occupa di orientare studenti o futuri studenti all'interno dell'Università Politecnica delle Marche. Ovviamente, ai fini del progetto, non è stato possibile fornire le indicazioni sulla base di tutte le possibili richieste degli studenti, ma si è deciso di focalizzare l'attenzione su un sottoinsieme di esse. Nello specifico, gli utenti potranno:

- chiedere informazioni sulla posizione delle aule
- chiedere informazioni sulle tasse e relative scadenze
- chiedere informazioni sulle facoltà presenti presso l'Università Politecnica delle Marche e sui relativi corsi di laurea
- ottenere i piani di studio e chiedere informazioni dettagliate sui singoli corsi proposti. Questa funzionalità, in particolare, è stata progettata e sviluppata solo per il corso di laurea in Ingegneria Informatica e dell'Automazione (sia triennale che magistrale).

Dalle funzionalità elencate sopra, è emersa l'esigenza di un dataset che permettesse al chatbot di estrarre le informazioni richieste dall'utente. Vista l'assenza di API già predisposte che potessero sopperire in maniera diretta alla questione, si è resa necessaria una prima fase di creazione manuale di tali dataset. Maggiori dettagli sui file creati e poi utilizzati dal chatbot sono riportati di seguito.

Aule.xlsx	
Polo	Polo universitario in cui l'aula è presente
Piano	Piano del polo in cui si trova l'aula
Aula	Nome dell'aula

Corsi.xlsx	
ID	ID del corso
Corso	ID del corso di laurea di appartenenza
Nome	Nome del corso
Anno	Anno di erogazione
Semestre	Semestre di erogazione
Tipologia	Tipologia del corso: a scelta (S) oppure obbligatorio (O)
CFU	Numero di CFU associati al corso
Settore	Settore di appartenenza del corso
Descrizione	Descrizione breve degli obiettivi del corso

Facolta_Dipartimento.csv	
Facolta	Facolta in cui il corso di laurea è erogato
ID	ID del corso di laurea
Nome	Nome del corso di laurea
Classe	Classe del corso di laurea
Laurea	Tipologia del corso di laurea: triennale o magistrale
Descrizione	Descrizione breve del corso di laurea

Le tabelle appena descritte, seppur destinate ai soli fini del progetto, sono state popolate con informazioni veritiere (e pubbliche) raccolte dal sito ufficiale di ateneo

1.2.1 Struttura del progetto

Una volta aver creato una directory di progetto ed aver attivato l'ambiente virtuale su cui mandare in esecuzione RASA, il framework mette a disposizione il comando *rasa init* per creare una struttura di progetto completa di tutto il necessario per poter intraprendere lo sviluppo.
Nella figura 1.1 è possibile avere un'idea dell'organizzazione dei file e delle cartelle di progetto.

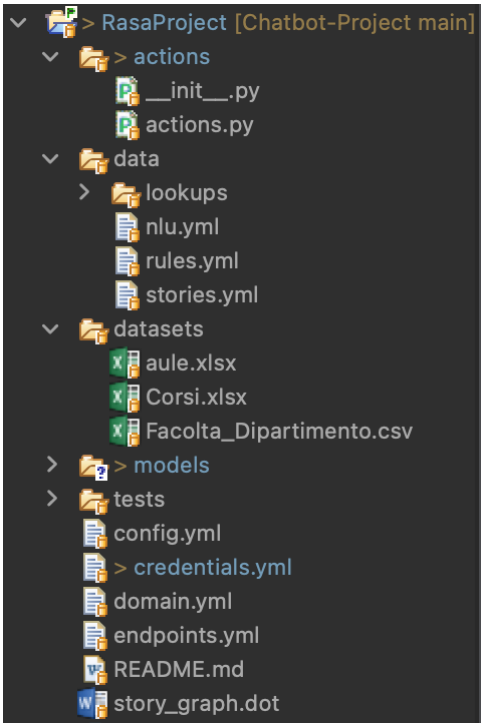

Figura 1.1: Struttura del progetto RASA per l'UNIVPM chabtot

Di seguito vengono commentati i file principali su cui si è concentrato il lavoro:

- **nlu.yml**: in questo file sono stati specificati i dati di addestramento per il chatbot. Nello specifico, gli intent con i relativi testi di esempio.
- **domain.yml**: file contenente tutto ciò che il chatbot deve sapere per poter operare. In particolare, è possibile trovare una lista di tutti gli intent che dovranno essere trattati con le eventuali entity utilizzate, gli slots per memorizzare informazioni necessarie all'elaborazione, le *utterences* con le risposte che il chatbot fornirà, la definizione di form ed infine la lista riportante le azioni.
- **stories.yml**: file contenente pattern di ipotetiche conversazioni, chiamati path, che il chatbot seguirà.
- **rules.yml**: contiene regole particolari che dovranno essere seguite dal chatbot.
- **actions.py**: script Python in cui sono definite tutte le classi che mappano le azioni riportate nel file *domain.yml* e che necessitano di una particolare elaborazione per costruire la risposta.

Capitolo 2

Facoltà e relative informazioni

In questo capitolo verranno esaminate le scelte e le strategie affrontate in merito allo sviluppo delle funzionalità che permettono al chatbot di rispondere a richieste relative a cosa gli studenti possono studiare presso l'Università Politecnica delle Marche e a maggiori dettagli sulle singole facoltà e sui corsi di laurea.

2.1 Facoltà

In una prima fase si è ragionato su ciò che un futuro studente potrebbe chiedersi quando si trova di fronte ad una scelta universitaria, ovvero cosa si può studiare presso un politecnico come quelle delle Marche. Questo permette all'utente di avere una idea molto generale riguardo ai possibili corsi e quindi, sulla base di quelle che sono le facoltà, potrebbero nascere in lui ulteriori domande per avere maggiori dettagli.
Per questo motivo, sempre a questo livello, si è pensato di gestire eventuali richieste di dettaglio riguardo le singole facoltà, facendo si che il chatbot in tali situazioni restituisca i vari corsi di laurea che vengono erogati, suddivisi per tipologia di laurea, ovvero triennale, magistrale, magistrale a ciclo unico, e ad orientamento specializzato (quest'ultimo solo nel caso di Ingegneria).
In questo modo lo studente riesce ad avere una consapevolezza maggiore rispetto all'inizio su ciò che egli stesso può studiare presso l'Università Politecnica delle Marche.

Fornita una descrizione generale di ciò che il chatbot può effettuare a questo livello, adesso si veda più nel dettaglio come tutto ciò può essere realizzato.
A tal proposito in Figura 2.1 è riportato un estratto di codice in cui sono definiti alcuni esempi per l'*intent* in questione.

```
- intent: find_info_faculties
  examples: |
    - puoi darmi informazioni su [ingegneria](facolta)?
    - vorrei avere informazioni su [agraria](facolta)
    - quali sono i corsi di laurea di [medicina e chirurgia](facolta)?
    - dimmi quali sono i corsi di laurea di [scienze](facolta)

- intent: find_studies
  examples: |
    - cosa posso studiare all'UNIVPM?
    - dimmi che cosa si studia all'UNIVPM
    - quali sono le facolta in cui posso studiare?
```

Figura 2.1: Intent ed esempi per la richiesta di informazioni sulle facolta

Ovviamente si osserva che l'intent *find_studies* è molto generico, in quanto l'utente si trova nella situazione più generica possibile dove vuole conoscere quali tipologie di facoltà ci sono presso l'Università Politecnica delle Marche, mentre nel caso dell'intent *find_info_faculties* ci si aspetta che venga riportata anche la facoltà di cui si vogliono ottenere delle informazioni.
Il successivo passaggio è stato quello di definire un'azione che fosse stata in grado di gestire la richiesta in questione. In realtà questo non si è reso necessario per l'intent *find_studies*, in quanto l'output poteva essere facilmente gestito tramite un *utter_response* con all'interno il testo di risposta di interesse. Per creare un'azione del genere, dentro il file *actions.py*, si è andati a creare la classe *ActionFindInfoFaculties*. In essa il metodo *run* è responsabile delle azioni da svolgere e brevemente la logica è la seguente:

1. Estrarre dallo slot *facolta* il valore corrispondente inserito dall'utente
2. Estrarre dal file *Facolta_Dipartimento.csv* i corsi di laurea corrispondenti alla facoltà inserita. Nel caso in cui non ci fosse nulla, viene gestito tale caso limite con un output opportuno che segnali ciò
3. Creare tante liste quante tipologie di laurea vi sono, ad esempio triennale, magistrale, etc.
4. Restituire in uscita l'output formattato

2.2 Corsi di Laurea

Come affermato in precedenza, il chatbot ha anche la possibilità di fornire chiarimenti e dettagli sui singoli corsi di laurea, restituendo per ognuno di essi una descrizione breve di quello che viene erogato, di quali sono gli obiettivi e le aspettative degli studenti.
Per gestire questa richiesta l'idea è stata di sfruttare le **form** che il framework Rasa mette a disposizioni. Infatti, esse permettono di raccogliere in maniera incrementale i dati, il che è un meccanismo che si adatta molto bene alla situazione in questione, visto che l'idea è quella di leggere prima la tipologia di laurea e poi il nome del corso.
Innanzitutto, in Figura 2.2 viene mostrato un estratto di codice che riporta alcuni esempi per l'intent in questione.

```
- intent: find_info_addresses
  examples: |
    - vorrei informazioni su un corso di laurea
    - posso avere informazioni su un corso di laurea?
```

Figura 2.2: Intent ed esempi per la richiesta di informazioni sui corsi di laurea

La form in questione è stata definita nel file *domain.yml*, come mostrato in Figura 2.3.

```
forms:
  find_info_addresses_form:
    required_slots:
      - laurea
      - indirizzo
```

Figura 2.3: Definizione form per informazioni su un corso di laurea

Come si può osservare alla form sono stati associati gli slot richiesti che l'utente dovrà poi riempire in fase di inserimento. Quindi, tali informazioni dovranno essere memorizzate e verranno poi utilizzate per delle azioni future e questo meccanismo di memorizzazione è reso possibile tramite gli *slot*, come si può osservare in Figura 2.4 .

```
laurea:
  type: text
  mappings:
  - type: from_text
    conditions:
    - active_loop: find_info_addresses_form
      requested_slot: laurea
indirizzo:
  type: text
  mappings:
  - type: from_text
    conditions:
    - active_loop: find_info_addresses_form
      requested_slot: indirizzo
```

Figura 2.4: Slot per form su un corso di laurea

Come si può osservare in questo caso viene reso esplicito il legame con i campi *required_ slot* che erano stati citati in precedenza.
Il successivo passaggio è stato quello di definire un'azione ad hoc per la form in questione, la quale brevemente effettua i seguenti passi:

1. Sulla base dei campi inseriti dall'utente estrae i valori dagli slot *laurea* e *indirizzo*
2. Dal file *Facolta_ Dipartimento.csv* si estrae il corso di laurea in questione. Se l'estrazione restituisce un risultato nullo viene segnalata una condizione di errore in cui l'utente ha inserito un nome di un corso di laurea che non esiste.
3. Si controlla che la tipologia di laurea sia *triennale*, *magistrale*, *magistrale a ciclo unico* o *orientamento professionale*. Se così non fosse viene segnalato una condizione di errore all'utente

4. Se i due controlli precedenti vanno a buon fine si estrae la descrizione relativa al corso di laurea in questione e lo si comunica in uscita.

Infine, l'ultimo passaggio è stato quello di inserire due *rules* che avessero permesso di regolare tutto questo meccanismo, ovvero l'attivazione della forma e la sua submit. L'estratto del codice in questione è riportato in Figura 2.5.

```
- rule: Activate form corsi
  steps:
  - intent: find_info_addresses
  - action: find_info_addresses_form
  - active_loop: find_info_addresses_form

- rule: Submit form corsi
  condition:
  - active_loop: find_info_addresses_form
  steps:
  - action: find_info_addresses_form
  - active_loop: null
  - slot_was_set:
    - requested_slot: null
  - action: action_find_info_addresses
```

Figura 2.5: Rules per form su un corso di laurea

Capitolo 3

Piano di studio ed informazioni sui singoli insegnamenti

In questo capitolo verranno esaminate le strategie, le scelte e le eventuali problematiche affrontate in merito allo sviluppo delle funzionalità che permettono al chatbot di rispondere a richieste di informazioni sul piano di studio e su specifici corsi.
Come anticipato nel capitolo 1, ai fini del progetto, questa sezione è riferita esclusivamente alla facoltà di Ingegneria Informatica e dell'Automazione.

3.1 Piano di studio

Ragionando su quelle che, mediamente, possono essere le ricerche più gettonate e gli iter più frequenti (tra gli stessi membri del gruppo di progetto) all'interno del sito di Ateneo, si è pensato che per uno studente possa risultare comodo avere un'idea immediata dei corsi che dovrà/potrà seguire in uno specifico periodo della propria carriera universitaria. Da qui l'idea di trattare i piani di studio.
Un piano di studio, per i nostri intenti, è caratterizzato da tre informazioni principali: la tipologia di laurea (triennale o magistrale), l'anno di corso e il semestre. Alla luce di ciò, si è cercato di mettere il chatbot nelle condizioni di poter rilevare tali informazioni fondamentali nel corso dell'interazione con l'utente ed, una volta memorizzate, poter stilare una lista degli insegnamenti previsti o opzionabili nel periodo scelto.

Ripercorrendo il tipico workflow di sviluppo in RASA, si è partiti innanzitutto dal file *nlu.yml*, introducendo un nuovo *intent* dedicato alla possibile richiesta di informazioni sul piano di studio. La figura 3.1 mostra un estratto di codice riportante alcuni esempi di training per l'*intent* in questione.

Come è possibile notare, a questo livello, l'intent è molto generico e non c'è traccia di eventuali parole chiave necessarie ad individuare un semestre specifico, di un certo anno, per una determinata tipologia di laurea. In effetti, ciò è dettato da una scelta ben precisa, il cui scopo è quello di condurre l'utente a fornire tali informazioni passo passo, in maniera ordinata, evitando di costringerlo a dover formulare una domanda che rispecchi un format piuttosto rigido.
A tal proposito, l'idea è stata quella di utilizzare uno dei pattern più comuni per effettuare

```
- intent: request_lista_corsi
  examples: |
    - vorrei conoscere il piano di studio
    - qual è il piano di studio?
    - quali corsi devo frequentare?
    - che corsi sono previsti?
    - quali corsi devo seguire?
    - che corsi ci sono?
    - vorrei sapere che corsi dovrò seguire
```

Figura 3.1: Intent ed esempi per la richiesta di informazioni sul piano di studio

il cosiddetto *slot filling*, cioè raccogliere più pezzi di informazione da un utente al fine di una qualche elaborazione, le **form**.
Sfruttando una form, infatti, è stato possibile "dilazionare" la richiesta dell'utente attraverso una serie di botta e risposta in cui, ad ogni step, egli è chiamato a specificare uno ed un solo *key point*.
La form per il piano di studio, come da regola, è stata specificata nel file *domain.yml*, secondo la struttura mostrata in figura 3.2. Alla form è stato associato un nome univoco e gli slots richiesti mappano quelli che possono essere pensati come i campi che l'utente dovrà compilare.

```
forms:

  lista_corsi_form:
    required_slots:
        - laurea_2
        - anno
        - semestre
```

Figura 3.2: Definizione della form per la richiesta di informazioni sul piano di studio.

Considerato che, trattandosi di un chatbot, la form non viene mostrata come un modulo statico da compilare autonomamente bensì viene presentata nell'arco di un dialogo, sono state create anche le tre *utterences* corrispondenti alle domande che il chatbot deve porre all'utente in ognuno degli steps, rispettivamente riguardo la tipologia di laurea, l'anno e il semestre desiderati.

Nel momento in cui si utilizza una form - inoltre - si intende memorizzare delle informazioni fornite dall'utente e necessarie ad azioni future, motivo per cui si è fatto uso degli *slots*. Gli slots definiti per la form in questione sono mostrati in figura 3.3.

```
slots:
  laurea_2:
    type: text
    mappings:
    - type: from_text
      conditions:
      - active_loop: lista_corsi_form
        requested_slot: laurea_2
  anno:
    type: text
    mappings:
    - type: from_text
      conditions:
      - active_loop: lista_corsi_form
        requested_slot: anno
  semestre:
    type: text
    mappings:
    - type: from_text
      conditions:
      - active_loop: lista_corsi_form
        requested_slot: semestre
```

Figura 3.3: Codifica degli slots necessari alla form sul piano di studio

Come è possibile notare, gli slots riportati fanno riferimento ai nomi utilizzati nel campo *required_slots* all'interno della definizione della form.
Oltre a ciò, sono da evidenziare alcune differenze rispetto agli slots trattati nei capitoli precedenti, i quali mappavano una *entity*. In primo luogo, la tipologia di mapping, in questo caso, è *from_text*, a specificare che lo slot in questione deve essere riempito direttamente con il testo inserito dall'utente; dopodiché, è presente una condizione di *active_loop* la quale dice che lo slot deve essere attivato quando viene mandata in esecuzione la form.

Il passo successivo ha visto la creazione di un'azione *ad hoc* che, sulla base delle preferenze raccolte tramite l'input dell'utente nel corso della form, fosse in grado di restituire, come risposta, la lista dei corsi tenuti, andando a selezionare le righe opportune nella tabella del file *Corsi.xlsx*.
Per creare un'azione del genere, si è andati ad inserire la classe *ActionCoursesList* all'interno del file *actions.py*. Il metodo *run* di tale classe è il responsabile delle operazioni da svolgere e, alla base, ha la seguente logica:

- innanzitutto, per mezzo del *Tracker*, viene estratto il contenuto degli slots "laurea_2", "anno", "semestre", cioè i dati forniti dall'utente in risposta alle domande della form;
- si procede ad un processing di tale contenuto, in relazione alle caratteristiche della tabella *Corsi*: ad esempio, l'attributo *Corso*, che discerne se un insegnamento è erogato durante il triennio o il biennio, presenta esclusivamente dei codici, "IT04" nel primo caso o "IM12" nel secondo; l'utente, invece, guidato dalla form, scriverà qualcosa di più naturale, una parola. Per questo motivo, il mapping dell'input dell'utente in uno dei due codici presenti nella tabella viene effettuato dietro le quinte dall'azione, sulla base del fatto che lo slot "laurea_2" contenga la parola "triennale"

o "magistrale" (se l'utente inserisce una parola diversa, dunque fuorviante in questo contesto, la variabile viene settata a *null*);

- successivamente, sfruttando la libreria *Pandas* di Python, viene creato un *dataframe* leggendo dal file *Corsi.xlsx* e vengono selezionate le righe, cioè gli insegnamenti, tali per cui gli attributi *Corso*, *Anno*, *Semestre* siano uguali, rispettivamente, agli slots "laurea_2", "anno", "semestre";

- si costruisce il testo di output, da passare successivamente al *Dispatcher*, che costituirà la risposta del chatbot. Tale testo contiene la lista degli insegnamenti filtrati secondo i vincoli di cui sopra; per ogni corso, inoltre, viene indicata la tipologia, cioè se è obbligatorio o opzionabile a scelta dallo studente;

- parallelamente a tutto ciò, sono gestiti i casi limite in cui il contenuto degli slots non è in grado di individuare alcuna riga nella tabella. In tal caso, il testo di output conterrà un messaggio di errore perché, probabilmente, l'utente avrà sbagliato a digitare o avrà inserito dati incoerenti (es: richiesta dei corsi previsti nel terzo anno della magistrale)

Una volta aver messo a punto l'azione appena descritta, l'ultimo step è stato quello di inserire due *rules* nell'omonimo file .yml che regolassero l'attivazione della form e l'evento scatenato dal submit di quest'ultima.
Per mostrare il setting di tali regole, e al tempo stesso avere un quadro del path che parte dalla richiesta di informazioni sul piano di studio e termina con la lista dei corsi come risposta del chatbot, passando dalla form, si faccia riferimento alla figura 3.4.

```yaml
- rule: Activate form piano studi
  steps:
  - intent: request_lista_corsi
  - action: utter_richiesta_info
  - action: lista_corsi_form
  - active_loop: lista_corsi_form

- rule: Submit form piano studi
  condition:
  - active_loop: lista_corsi_form
  steps:
  - action: lista_corsi_form
  - active_loop: null
  - slot_was_set:
    - requested_slot: null
  - action: action_courses_list
```

Figura 3.4: Codifica delle rules necessarie per la gestione della form sul piano di studio

3.2 Informazioni sul singolo insegnamento

Immaginando di proseguire in un ipotetico discorso con il chatbot, dopo aver ottenuto la lista dei corsi previsti sulla base di tipologia di laurea, anno e semestre desiderati, si è pensato di andare a fondo sui singoli corsi.
L'idea, dunque, è stata quella di dare all'utente la possibilità di chiedere al chatbot informazioni su uno specifico corso, una descrizione di esso.

Il punto di partenza, come di consueto, è stato il file *nlu.yml*, con la scrittura di un *intent* che esprimesse la volontà dell'utente di richiedere informazioni su un corso a sua scelta. L'intent *find_info_courses* e i relativi esempi di training sono riportati nell'estratto di codice mostrato in figura 3.5.

```
- intent: find_info_courses
  examples: |
    - puoi darmi informazioni su [analisi matematica 1](corso)?
    - vorrei avere informazioni sul corso [elettrotecnica](corso)
    - vorrei avere informazioni su [linguaggi di programmazione](corso)
    - dimmi qualcosa su [data science](corso)
    - dammi informazioni su [fisica](corso)
    - ho bisogno di informazioni riguardo [controlli automatici](corso)
    - avrei bisogno di info riguardo [fisica generale 2](corso)
    - vorrei sapere qualcosa sul corso [programmazione mobile](corso)
```

Figura 3.5: Codifica dell'intent per la richiesta di informazioni su un insegnamento specifico

Come è possibile notare dalla figura 3.5, gli esempi di richieste utilizzati per l'addestramento del chatbot presentanto la particolare sintassi che evidenzia l'esistenza di una *entity*. Quando si richiedono informazioni circa un corso nello specifico, sostanzialmente, ci si sta riferendo ad una particolare istanza della classe di oggetti "corso".
Ovviamente, però, non potevano essere forniti tanti esempi quanti sono i singoli insegnamenti erogati per il corso di laurea di Ingegneria Informatica e dell'Automazione (57 insegnamenti), dunque è stata utilizzata una *look-up table*, da cui il chatbot può prelevare le istanze dell'*entity* "corso".
Un estratto della *look-up table* appena discussa, presente all'interno del file *corso.yml* nella cartella *lookups*, è mostrato in figura 3.6.

```
version: "3.0"
nlu:
  - lookup: corso
    examples: |
      - algebra e logica
      - algebra lineare e geometria
      - analisi complessa
      - analisi matematica 1
      - analisi matematica 2
      - analisi numerica
      - automazione industriale
      - calcolatori elettronici e reti di calcolatori
      - calcolo delle probabilità e statistica matematica
      - controlli automatici
      - economia dell'impresa
      - elementi di elettronica
      - elettromagnetismo per la trasmissione dell'informazione
      - elettrotecnica
      - fisica generale 1
```

Figura 3.6: Look-up table relativa all'*entity* "corso"

Il nome dell'insegnamento che l'utente scriverà, dunque, deve essere individuato e memorizzato dal chatbot affinché, in un secondo momento, sia possibile la sua ricerca all'interno del dataset relativo ai corsi. Di conseguenza, passando al file *domain.yml*, è stato inserito uno slot che mappasse l'*entity* "corso" come riportato in figura 3.7.

```
slots:
  corso:
    type: text
    mappings:
    - type: from_entity
      entity: corso
```

Figura 3.7: Slot in cui si memorizza il nome del corso di cui l'utente vuole avere informazioni

Una volta aver fatto ciò, è stata predisposta una classe *ActionFindInfoCourses* all'interno del file Python dedicato alle azioni, che effettua le seguenti elaborazioni:

- attraverso il *Tracker*, viene memorizzato il contenuto dello slot "corso" in un'opportuna variabile;
- sfruttando la libreria *Pandas*, si va a leggere il file *Corsi.xlsx* contente le informazioni relative ai corsi, creandone un dataframe;
- dopo aver inizializzato una lista vuota, si procede alla scansione delle righe del dataframe confrontando, di volta in volta, il nome dell'insegnamento scritto dall'utente (recapitato dallo slot) e il valore dell'attributo "Nome" del dataframe (i nomi dei corsi nella tabella sono univoci). Se c'è corrispondenza allora il nome e la descrizione di quell'insegnamento vengono inseriti in append alla lista altrimenti si prosegue con il ciclo for;
- si controlla che la lunghezza della lista sia pari ad 1 (è stato trovato il corso corrispondente al nome inserito dall'utente), dunque il chatbot risponderà con la descrizione di quel corso.

Quella appena discussa è la funzionalità di base dell'azione. In realtà, in un secondo momento, sono state formulate più ipotesi sul possibile comportamento dell'utente ed, in effetti, è sembrato troppo stringente partire dal presupposto che un utente scriva sempre il nome esatto come riportato sul dataset.
Se, ad esempio, l'utente richiede informazioni sul corso "Manutenzione Preventiva" anziché digitare l'intero nome "Manutenzione Preventiva per l'Automazione Intelligente e la Robotica", potrebbe risultare frustrante non ricevere risultati.
A tal proposito si è pensato di "rilassare" il confronto tra il nome inserito dall'utente e il nome presente nel dataset: durante la scansione delle righe della tabella, se c'è corrispondenza esatta tra i nomi del corso allora si esce immediatamente dal ciclo e viene restituito quell'unico risultato, altrimenti il corso viene selezionato anche se il nome scritto dall'utente è solamente in parte contenuto nel nome presente nella tabella.
A questo punto, però, si è posto un ulteriore problema, ossia quello delle molteplici corrispondenze. Se lo studente, infatti, volesse avere informazioni su "Analisi", lo script python selezionerebbe più righe della tabella: "Analisi Matematica 1", "Analisi Matematica 2", "Analisi Complessa" ... In questi casi, dunque, il chatbot restituisce l'elenco di corsi cui l'utente potrebbe riferirsi, chiedendogli successivamente di riformulare la richiesta con uno di quegli insegnamenti trovati, eliminando definitivamente l'incertezza. Tale eventualità è mostrata nella figura 3.8, in cui la risposta del chatbot è stata testata tramite la shell di Rasa.

```
Bot loaded. Type a message and press enter (use '/stop' to exit):
Your input ->  ciao
Ciao! Come posso esserti d'aiuto?
Your input ->  puoi darmi informazioni su analisi?
Ho trovato più corrispondenze. Riformula con uno dei seguenti corsi:
- Analisi Complessa
- Analisi Matematica 1
- Analisi Matematica 2
- Analisi Numerica
Your input ->
```

Figura 3.8: Risposta nel caso in cui l'utente richiede informazioni su un corso per cui vengono trovate più corrispondenze nel nome.

In ultima istanza, la figura 3.9 mostra la story creata per dar luogo al soddisfacimento della richiesta di informazioni circa uno specifico corso.

```
- story: find_info_courses path
  steps:
  - intent: greet
  - action: utter_greet
  - intent: find_info_courses
    entities:
    - corso: elettrotecnica
  - action: action_find_info_courses
```

Figura 3.9: Story con il path che porta il chatbot a rispondere su informazioni relative a singoli corsi. Nota: l'istanza specifica per l'entity utilizzata dall'intent *find_info_courses* è puramente esemplificativa e potrebbe essere sostituita da qualsiasi altro nome presente nella look-up table relativa al corso.

Capitolo 4

Informazioni sulle aule e sulle scandenze

4.1 Aule

In questo capitolo viene affrontata la funzionalità che permette ad un utente di recuperare informazioni sulla posizione delle aule della facoltà di ingegneria. Il file csv che funge da database *Aule.csv*, come visto nella parte introduttiva, si compone di tre indicazioni che sono il polo di appartenenza dell'aula, il piano con un riferimento ad alcuni punti noti in modo che l'utente possa orientarsi meglio su alcune aule più remote e infine il nome stesso dell'aula. Descritta in generale la funzionalità si veda ora l'implementazione nel dettaglio a partire dal file *nlu.yml* dove viene introdotto un nuovo *intent* denominato *find_info_aula*, mostrato in Figura 4.1, il quale riporta alcuni esempi di frasi che un utente può scrivere al bot per recuperare l'informazione richiesta. Il bot riconosce l'istan-

```
- intent: find_info_aula
  examples: |
    - dove si trova l'aula [145/1](aula)
    - a che piano è la [155/10](aula)
    - a che piano è l'aula [a9/a10](aula)
    - dove trovo l'aula [en1](aula)
    - dove trovo una [sala studio](aula)
    - a che piano è il [salone polifunzionale](aula)
    - dove sta il [salone polifunzionale](aula)
    - dove si trova la [biblioteca](aula)
    - dove si trova il [bar](aula)
    - a che piano è il [bar](aula)
    - dove si trova l'[aula magna](aula)
```

Figura 4.1: Intent ed esempi per la richiesta di informazioni sulle aule

za aula grazie all'utilizzo della entity *(aula)*. Una entity rappresenta infatti una classe di oggetti che l'NLU deve riuscire a capire dall'input dell'utente. Come da workflow Rasa l'entity è stata dichiarata nel file *domain.yml* e ad essa è stata associata una *lookup table*, mostrata in Figura 4.2, contenente tutti i nomi delle aule della facoltà di ingegneria. A questo punto è stata definita la action *action_find_info_aula* nel file *actions.py*. Tale azione cattura l'entity inserita dall'utente grazie al *tracker* che la va a recuperare nella *slots* apposita. La slots viene definita nel file *domain.yml* e contiene il riferimento alla

```
nlu:
  - lookup: aula
    examples: |
      - 145/1
      - 140/1
      - 140/2
      - 140/D3
      - 140/3
      - 140/D4
      - 140/D5
      - 145/1
      - 145/2
      - 145/3
      - 145/G1
      - 145/G2
      - 145/4
      - 145/5
      - 150/1
      - 150/2
      - sala studio
      - 155/D2
```

Figura 4.2: Lookup table contenente i nomi delle aule

entity aula. Fatto ciò la action scandisce tutte le righe del file Aule.csv cercando una corrispondenza nella colonna 'Aula' contenente appunto il nome dell'aula. Trovato il riscontro, il messaggio che viene formulato in output è formato dal nome del complesso di appartenenza dell'aula e il piano. Di seguito 4.3 viene riportato un esempio del messaggio in output dell'action.

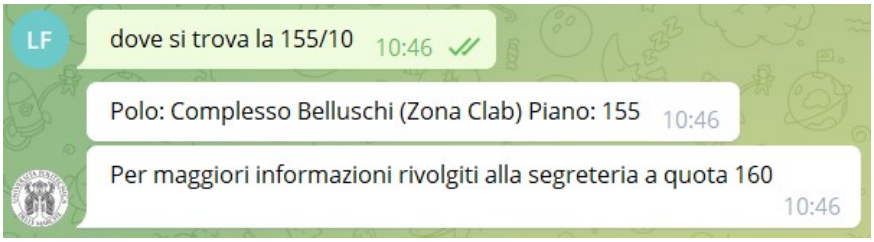

Figura 4.3: Esempio action_find_info_aula

Come si può notare dalla figura 4.3 dopo il messaggio relativo alla posizione dell'aula richiesta dall'utente è presente un ulteriore messaggio che di default viene stampato dopo ogni richiesta per fornire un aiuto addizionale all'utente. Questo avviene grazie alla *FollowUpAction* che permette di specificare una azione che deve essere eseguita dopo la action_find_info_aula; in particolare qui viene stampato un messaggio implementato tramite un *utter* dichiarato nel file domain.
Infine è stata aggiunta una rule 4.4 nell'apposito file rules.yml dove vengono descritti dei pezzi di conversazione che devono seguire sempre lo stesso path. In questo caso viene specificato che all'attivazione dell'intent *find_info_aula* deve corrispondere l'attivazione della action_find_info_aula.

```
- rule: Restituisce informazioni sulle aule
  steps:
  - intent: find_info_aula
  - action: action_find_info_aula
```

Figura 4.4: Rule per informazioni sulle aule

4.2 Tasse e Contributi

Un'altra funzionalità che è stata implementata è quella relativa alla possibilità di un utente di chiedere al chatbot informazioni relative alle tasse o più in generale su scadenze e contributi. Anche in questo caso si è partiti dal file *nlu.yml* dove è stato inserito l'intent rappresentato in Figura 4.5 *find_ tasse*.

```
- intent: find_tasse
  examples: |
    - quando scadono le [tasse](tax)
    - puoi dirmi la [scadenza](tax) delle tasse
    - quando scade la prima rata[tax]
    - quando bisogna pagare le rate[tax]
    - quanto vale la prima rata[tax]
    - quando scade la prima [tassa] universitaria
    - quando scade la seconda [tassa] universitaria
```

Figura 4.5: Intent ed esempi per la richiesta informazioni sulle tasse

In maniera analoga alla precedente funzione, è stata definita una entity *tax* contenente le varie istanze (parole del tipo tasse, rate, scadenze, etc), definite nell'apposita lookup table, che l'utente può richiedere al chatbot. In questo caso non è necessario definire una action nel file actions.py poichè il messaggio di risposta del chatbot è un semplice testo che può essere semplicemente definito attraverso l'utilizzo di un utter. Il path da seguire viene poi specificato come una regola mostrata in Figura 4.6 nel file *rules.yml*.

```
- rule: Restituisce informazioni sulle tasse
  steps:
  - intent: find_tasse
  - action: utter_response_tax
  - action: utter_more_tax_info
```

Figura 4.6: Rule per informazioni su tasse

I due utter definiti nella rule fanno riferimento in ordine di dichiarazione al messaggio che il chatbot restituisce in output contenente scadenze e informazioni sulle tasse da pagare nell'anno corrente. Il secondo utter invece contiene un link che fa riferimento alla pagina dedicata alle tasse e ai contributi del sito dell' UNIVPM così che l'utente abbia la possibilità di ottenere informazioni ancora più dettagliate.

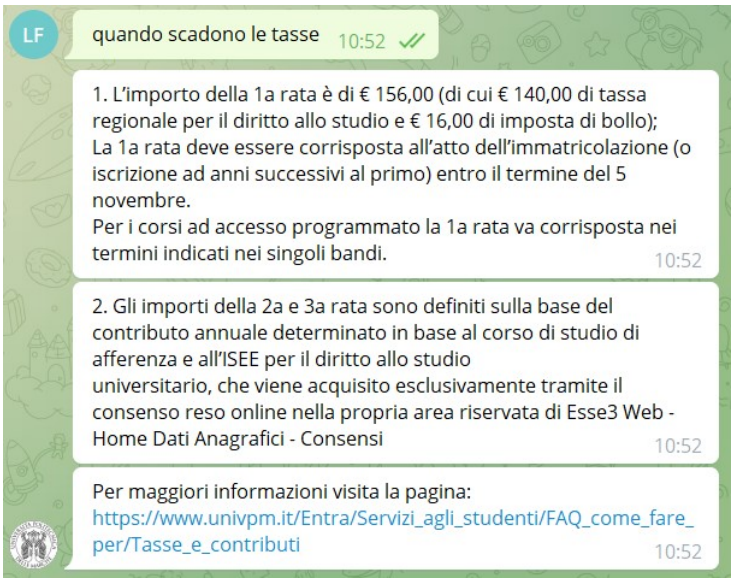

Figura 4.7: Richiesta dell'utente riguardo informazioni sulle tasse

Capitolo 5

Connessione a Telegram e Test

L'ultima fase del progetto ha previsto il collegamento del chatbot ad una piattaforma di messaggistica istantanea, in modo tale da renderlo disponibile agli utenti attraverso un'interfaccia moderna e soprattutto accedendo semplicemente da smartphone.
La piattaforma utilizzata a tal proposito è stata Telegram, scelta dettata sostanzialmente dalla facilità di caricamento di chatbot mediante una procedura guidata interna all'applicazione stessa e dalla possibilità di interconnessione con il servizio *ngrok*, responsabile di mandare in esecuzione il chatbot su server locale. In Figura 5.1 è riportata la configurazione necessaria all'interno del file *credentials.yml* per effettuare il deploy su Telegram.
In relazione alla figura, gli elementi settati sono i seguenti:

```
telegram:
  access_token: 5162109709:AAFdXc26DSbJVIKsMuKDSufITjqXXkS31fU
  verify: UNIVPMbot
  webhook_url:  "https://7845-87-20-68-191.ngrok.io/webhooks/telegram/webhook"
```

Figura 5.1: Configurazione bot telegram

- *access_token*: Token che si è ottenuto mediante la procedura guidata di deploy su Telegram tramite il bot *BotFather*
- *verify*: nome del bot in questione assegnato in fase di deploy su Telegram
- *webhook_url*: endpoint che permette alla macchina locale e al chatbot di comunicare. Tale endopoint è stato realizzato mediante *ngrok*.

5.1 Testing

A questo punto vengono riportati una serie di screen realizzati durante una simulazione del funzionamento del chatbot su Telegram, così da mostrare e puntualizzare i risultati discussi nei capitoli precedenti.

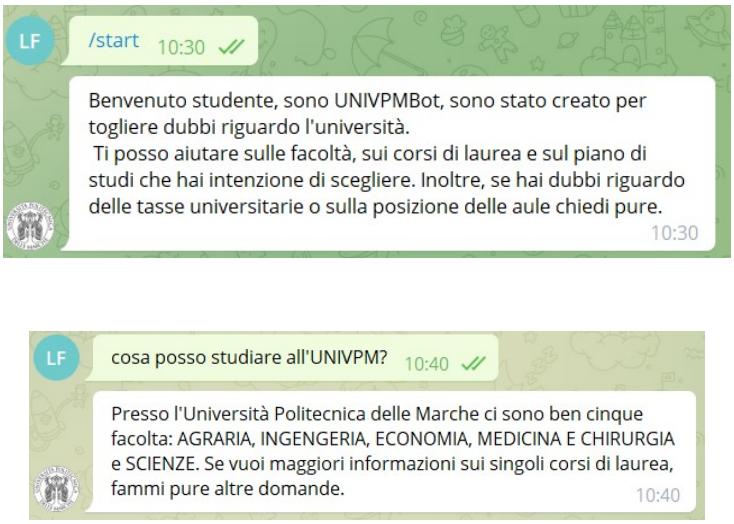

Figura 5.2: Richiesta informazioni sulle facoltà dell' UNIVPM

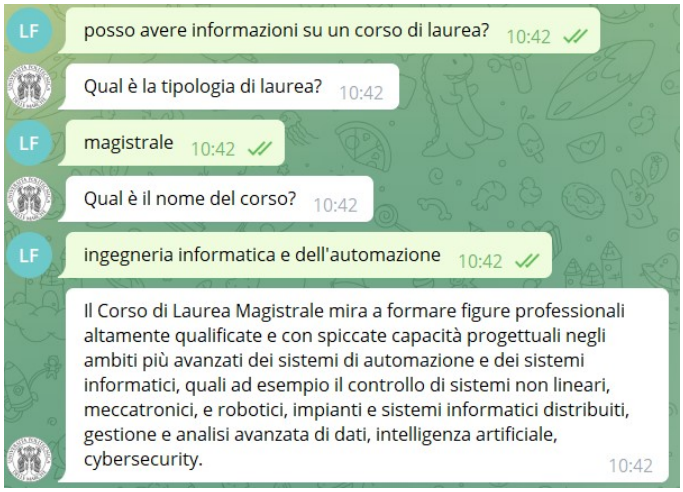

Figura 5.4: Richiesta riguardo uno specifico corso di laurea

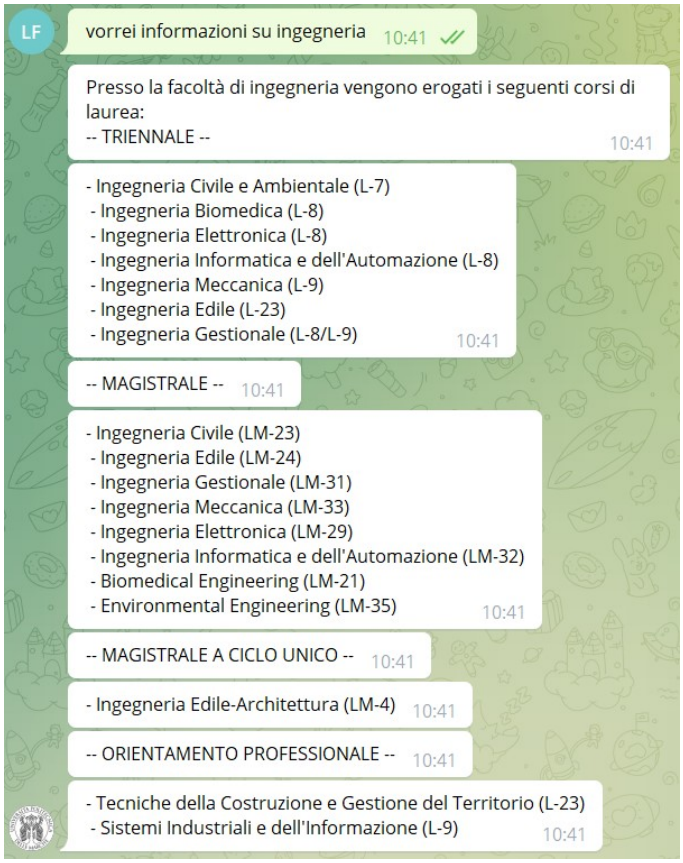

Figura 5.3: Richiesta su una specifica facoltà

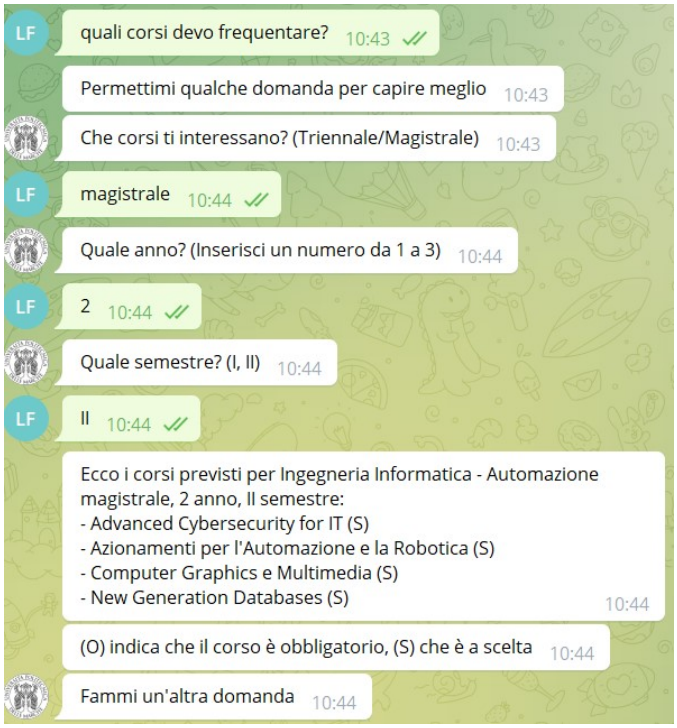

Figura 5.4: Richiesta di informazioni sui corsi da seguire (piano di studio)

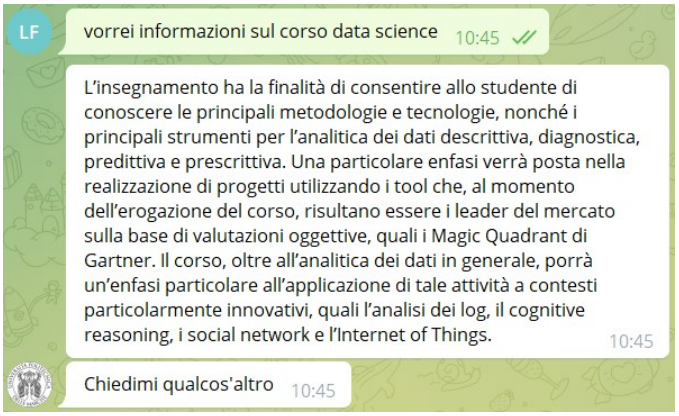

Figura 5.5: Richiesta di informazioni su un insegnamento specifico

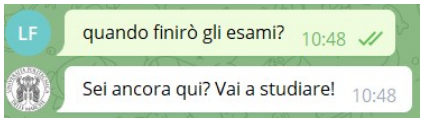

Figura 5.6: Risposta sarcastica

5.2 Sviluppi Futuri

In merito a quanto visto finora le funzionalità implementate del chatbot sono rivolte principalmente alla sola facoltà di ingegneria, sia per quanto riguarda la posizione delle aule sia per le informazioni dettagliate sui corsi di studio, quest'ultima funzione solo per ingegneria informatica e dell'automazione. Uno dei possibili sviluppi futuri sarebbe proprio quello di connettere il chatbot ad un vero database o ad una API, a differenza degli attuali file csv, ed espandere le funzionalità a tutte le facoltà e le sedi dell'UNIVPM così che il chatbot possa essere fruibile a tutti gli studenti.