I chatbot: framework di sviluppo e caso di studio

Progetto Ingegneria Informatica

Paolo Roncaglioni Stefano Sanitate

6 marzo 2018

Supervisors

Maristella Matera Vittorio Zaccaria Florian Daniel

Indice

1 Introduzione

Questo documento di specifica descrive lo stato dell'arte riguardo la tecnologia dei ChatBot e dei framework ad essi dedicati. Raccolte quindi informazioni su servizi disponibili e stato dell'arte abbiamo deciso di concretizzare la ricerca in una demo di un Bot che risponde alle esigenze di uno studente del Politecnico di Milano. Dopo una prima parte in cui vengono descritti in dettaglio quali test abbiamo eseguito prima di cominciare la progettazione vera e propria del bot, si passa a una stesura della specifica e formalizzazione di alcuni concetti chiave naturalmente appresi durante la prima fase di rassegna dei framework di sviluppo. La descrizione della specifica é iniziata con il delineare lo scopo del bot, specificatamente delineando le funzionalità e l'utenza. In un primo momento si è pensato di rivolgere il bot ad un'utenza di pendolari, ma ci siamo resi conto che tale servizio è già esistente (e.g. bot orari Trenitalia). Dopo aver quindi analizzato varie possibilitá di scelta delle funzionalitá del bot, abbiamo infine pensato di progettarne uno a tema Politecnico, che racchiuda un insieme di semplici tool per lo studente (o per il professore) nella gestione della propria vita universitaria.

2 Analisi dei framework di sviluppo

Lo scopo di questa sezione é di esplorare il mondo dei framework giá esistenti per trovare la soluzione piú adatta ai nostri scopi. Pertanto abbiamo pensato che il modo migliore per affrontare il problema fosse quello di progettare un chatBot di Test, e implementarlo in ciascuna di queste piattaforme. Esso quindi ci aiuterá con la selezione del nostro ambiente di lavoro in base a dei parametri illustrati sotto. É essenziale che sia un bot semplice, ma con funzionalitá che siano correlate alla costruzione del nostro progetto in specifica, come ad esempio l'uso di webhook.

2.1 Requisiti del bot di test

- Interazione basilari con un'API web.
- Riconoscimento domande al bot, piú domande poste in modo diverso devono essere riconosciute (e.g. com'è il tempo domani a Milano? = che tempo fa domani a Milano?).
- Riconoscimento di semplici entitá date dall'utente (e.g. luogo, ora, data, nome, ...).

2.2 Parametri di valutazione

Questi rappresentano i parametri usati per valutare i framework analizzati. Essi comprendono sia i componenti di valutazione della costruzione del bot, sia

quelli piú generali della piattaforma che stiamo usando. La valutazione è a nostra discrezione, ovviamente seguita da una minima descrizione delle procedure svolte.

Bot

- *Facilitá di progettazione*: quante conoscenze e approfondimenti sono necessari per creare il bot.
- *Tempo di implementazione*: quanto tempo impieghiamo per creare il bot finito e funzionante.
- *Profondità*: l'insieme di strumenti aggiuntivi che ci sono forniti dall'ambiente di lavoro (training, analisi, ...) che permettono uno sviluppo piú approfondito del bot.
- *Persistenza della memoria*: possibilitá (e facilitá) di memorizzare certi dati dell'utente per un certo periodo di tempo.

Framework

- *Pricing strategies*: prezzo e fornitori.
- *Piattaforme*: Su quali piattaforme di interazione (Telegram, Slack, ...) posso fare il deployment.
- *Disponibilitá di SDK*: utili allo sviluppo, non enormemente influenti in questo test.

2.3 Descrizione del bot di test

Il tipo di bot(test) che alla fine abbiamo scelto di portare avanti è una semplice implementazione di chatbot "Che tempo fa?", cioè data una localitá (milano, monza, ..) e un giorno (oggi, domani, 31/12/2017, ..) restituisce il tempo meteo della localitá scelta nella data scelta. Questo dovrebbe permetterci di esplorare tutti i parametri sopra descritti. La scelta è ricaduta su questo tipo di bot soprattutto grazie alla disponibilitá di una documentazione passo-passo da parte del frameWork DialogFlow dell'implementazione del bot descritto sopra (presente nella repository di github di riferimento).

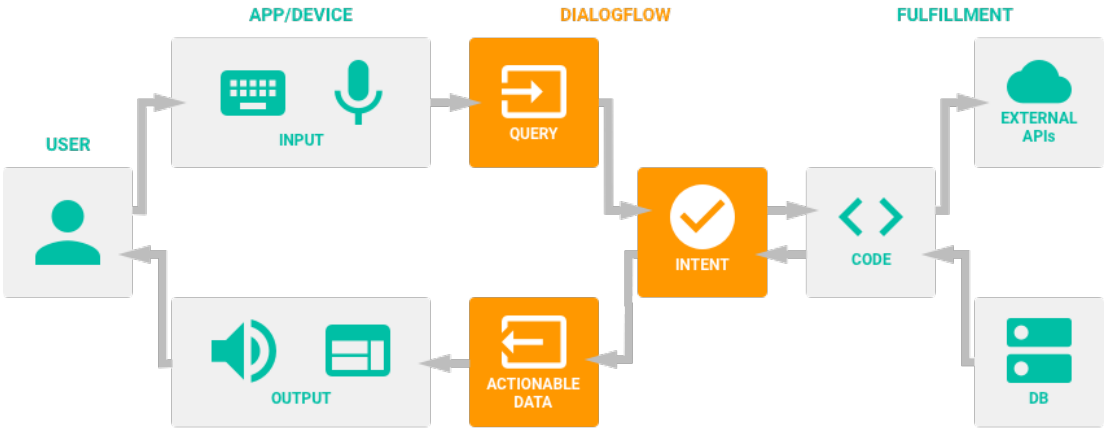
2.4 Sviluppo dei bot con i framework analizzati

In questa parte andiamo a descrivere la nostra esperienza con le 8 (su 25-30 di base, elencati in **appendice D**) piattaforme individuate tramite un'accurata ricerca. Abbiamo escluso tutte quelle piattaforme per il quale non è

stato possibile reperire il prezzo o che proponevano solo Trial o Demo (tipicamente pensate per aziende) o quelle che pensiamo non abbiano gli elementi base per il test. Per l'hosting e il deployment del codice (funzione weatherWebhook) che è servito da base per l'interazione con API esterne, abbiamo usato la piattaforma Google Cloud Project, usufruendo del servizio gratis per 1 anno. Per quanto riguarda l'API key, abbiamo sfruttato quella del sito `https://developer.worldweatheronline.com`, che permette di avere la premium key per un periodo di trial di 60 giorni (500 calls per minuto). Una parte su cui abbiamo avuto inizialmente difficoltá è stato lo studio del servizio di Google Cloud Platform, la cui configurazione si è rivelata più complicata del previsto.

2.4.1 DialogFlow

Implementazione La piattaforma permette di creare degli "agent", contenitori che servono a raccogliere i vari pezzi di riconoscimento delle risposte ("intent"). All'interno di un agent possiamo creare piú intent, cosicché la struttura finale risultante possa risultare anche abbastanza complessa. Piú intent possono essere concatenati creando un contesto. All'interno di un intent possiamo creare un processo di riconoscimento della domanda data dall'utente, preimpostando una risposta (testuale e non). All'interno del set di domande che l'utente puó scrivere, si possono settare dei "parameter" (gestione quasi automatica del parsing, possiamo comunque scegliere alcuni aspetti del json) che verrebbero quindi usati per la generazione delle risposte. Il riempimento dei parametri puó essere settato per essere preso all'esterno della piattaforma (usando un webHook) in modo molto semplice, inviando e ricevendo sempre un file in formato json. Detto questo è quindi bastato settare un paio di domande e risposte negli intent che abbiamo creato, inserire il giusto indirizzo di webhook e il bot è stato creato e testato su console e sul servizio di chat Telegram.



Dialogflow in breve

Valutazione

- *Facilitá di progettazione*: **9**
  molto semplice e intuitivo, senza necessità di scrivere codice
- *Tempo di implementazione*: **9**
  2 ore (escluso il setup di Google Cloud Platform)
- *Profondità*: **8**
  permette di creare facilmente contesti di riconoscimento, presenza di azioni di trigger ed eventi facilmente configurabili, sezione "Training" attivabile, riconoscimente delle domande e parametri non prima specificati
- *Persistenza della memoria*: **7**
  permette di tenere traccia di parametri (settando un lifespan) da poter riusare nel giusto contesto, non abbiamo svolto ulteriori ricerche
- *Pricing strategies*: **9**
  piattaforma completamente gratuita per la progettazione e per il deployment con un limite di richieste al minuto, superate le quali si ha l'opzione di fare un upgrade a pagamento del proprio piano
- *Piattaforme*: **9**
  vasta estensione, comprende buona parte dei servizi piú usati ad oggi di chat (Messenger, Telegram, Twitter, ...) e non (Slack, Google Assistant, Cortana,..)
- *Disponibilitá di SDK*: **10**
  presente buona parte dei linguaggi di programmazione piú usati

Commento personale DialogFlow (ex API.ai) é un ottimo candidato al framework che useremo nel nostro progetto. Semplice e intuitivo, permette di creare bot di varia natura e complessità, se usato nel pieno potenziale, senza richiedere troppo sforzo per il parsing di risposte/domande. Offre molte funzioni interessanti che non abbiamo approfondito, ma che risulterebbero utili al nostro sviluppo.

2.4.2 FlowXo

Implementazione FlowXo è il primo servizio nel quale per prima viene chiesta la piattaforma per la quale sviluppare il bot. Abbiamo scelto Telegram: il sito si appoggia a @BotFather per creare e autenticare il proprio bot. Ad un proprio bot è possibile associare un flow, nel quale è possibile assegnare una serie di azioni (tra cui una richiesta http, nel nostro caso).

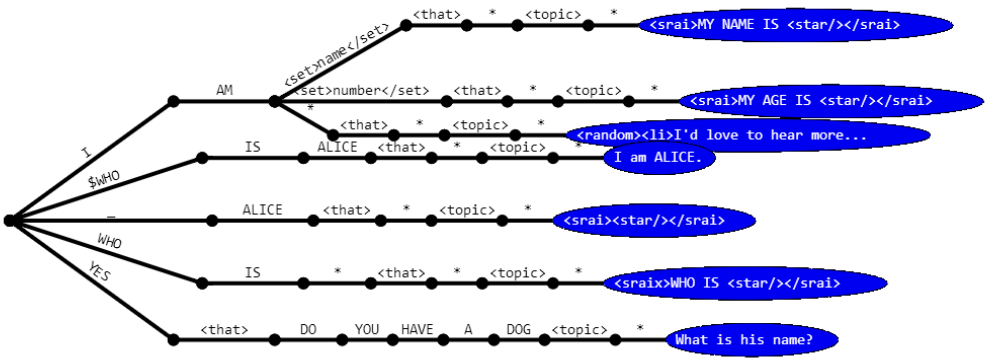
Valutazione

- *Facilitá di progettazione*: **9**
  a differenza di altri servizi FlowXo (come suggerisce il nome) focalizza la propria attenzione sul concetto di flow che nel nostro caso è stato piuttosto comodo
- *Tempo di implementazione*: **9**
  2 ore
- *Profondità*: **8**
  presenta tutte le funzioni base di un servizio del genere
- *Persistenza della memoria*: **7**
  permette di tenere traccia di parametri (settando un lifespan) da poter riusare nel giusto contesto, non abbiamo svolto ulteriori ricerche
- *Pricing strategies*: **5**
  si parte gratuitamente per un periodo di prova, per poi passare ad un costo mensile non irrisorio che comprende 5000 interazioni mensili, 15 bot e 3 mesi di log
- *Piattaforme*: **4**
  limitato a Telegram
- *Disponibilitá di SDK*: n/a

Commento personale L'utilizzo massiccio del concetto di flow è stato interessante rispetto alle altre soluzioni, ma abbiamo preferito optare per un approccio più libero. Ciò non toglie che tale approccio possa essere più user friendly e intuitivo. Non ci è piaciuto il pricing, avremmo preferito una soluzione free da eventualmente arricchire, piuttosto che un periodo di prova.

2.4.3 PandoraBots

PandoraBots offre un servizio di sviluppo di bot molto profondo e personalizzabile. Il tutto è basato su **AIML (Artificial Intelligent MarkUp Language)**, un'estensione di XML che si è costretti ad imparare se si vuole usare questa piattaforma. Difatti il servizio offerto è quasi solamente dato da interazione con Editor direttamente sul sito.



AIML Tree view

Implementazione In prima fase è necessario creare un bot e dargli un nome. All'interno possiamo trovare 5 pagine principali, rispettivamente Bot info (modifica informazioni del bot), Train (parlare col bot e cambiare risposte senza entrare in editor), Files (editor di AIML ottimizzato), Log (dove è possibile vedere le singole interazioni col bot) e ClubHouse (possibile pubblicare il proprio bot per farlo parlare con la comunitá). Senza andare troppo nel dettaglio con la descrizione del linguaggio AIML (che è stato necessario apprendere), si è dovuto metter mano ai tag in stile XML. <Pattern> (cosa dice l'utente), <Template> (la risposta desiderata a un pattern), <star/> (per "catturare" un input utente) sono solo alcuni dei possibili strumenti che offre questo linguaggio. È possibile creare dei contesti per il quale è possibile richiamare altri template, oppure creare argomenti in cui è possibile "entrare" per dare delle risposte specifiche, appunto in base all'argomento. Il tutto condito dalla possibilitá di aggiungere recursioni o condizioni come un normale linguaggio di programmazione. Queste funzioni base sono servite per creare un semplice bot che riconoscesse localitá e data di una domanda sul tempo, rimandando a un contesto e creando un argomento per chiedere singolarmente data o localitá. Grave mancanza è l'impossibilitá o l'estrema difficoltá di usare del contenuto dinamico, limitato a prendere stringhe da file.

Valutazione

- *Facilitá di progettazione*: **5**
  È stato necessario apprendere AIML per poter sviluppare un bot
- *Tempo di implementazione*: **7**
  Una volta apprese le basi di AIML non è stato particolarmente lungo il tempo di sviluppo, 3-4 ore probabilmente
- *Profondità*: **8**
  Al contrario di altre piattaforme, con PandoraBots è possibile avere il

pieno controllo del proprio bot. Invece scarno per quanto riguarda le funzionalitá accessorie

- *Persistenza della memoria*: **10**
  Grazie alle peculiaritá di AIML è possibile operare come un vero linguaggio, disponendo appunto di costanti e variabili (globali/locali) che si possono manipolare a piacimento

- *Pricing strategies*: **8**
  Lo sviluppo è totalmente gratuito, ma dopo le 1000 interazioni al mese, scatta il tassametro. Possibilitá di esportare il proprio bot localmente e metterlo sul proprio sito

- *Piattaforme*: **8**
  Molte piattaforme supportate come per gli altri Framework (compreso Telegram)

- *Disponibilitá di SDK*: **7**
  Disponibili i soliti Java, Node.js, Python per semplificare l'interazione con l'API di PandoraBots

Commento personale Di per sé, PandoraBots offre degli strumenti approfonditi per lo sviluppo "dell'intelligenza" del nostro bot, essendo costretti a progettare totalmente da zero il riconoscimento degli intenti e il flow in generale. La limitazione piú importante è senza dubbio l'impossibilitá di poter sfruttare degli elementi dinamici qualsiasi, o anche solo richiamare API esterne, feature per noi indispensabile.

2.4.4 GupShup.io

Implementazione Alla creazione del bot viene chiesto di scegliere innanzitutto se voler manualmente programmare il proprio bot oppure utilizzare un'interfaccia grafica. Nel primo caso si viene reindirizzati ad un IDE, nel quale è possibile partire da zero oppure scegliere un template di partenza e sviluppare quindi su una base già scritta. Nel secondo caso si viene guidati attraverso una GUI piuttosto user friendly, nel quale è possibile sviluppare un bot attraverso un metodo grafico, seppur con funzioni limitate alle basi. Ho ritenuto opportuno testare la prima opzione: qui ho avuto la possibilità di configurare un bot che usasse le funzioni già implementate con DialogFlow con l'utilizzo di un access token. In seguito è stato possibile testare il bot attraverso un proxybot proprietario di GupShup.

Valutazione

- *Facilitá di progettazione*: **8**
  è opportuno sottolineare che il sito offre due esperienze di progettazione completamente opposte: da un lato viene data completa gestione allo

sviluppatore, con conseguenti maggiori opzioni, mentre dall'altro viene offerta un'esperienza radicalmente più semplice ma comunque efficace per bot meno complessi.

- *Tempo di implementazione*: **9**
  il giudizio è chiaramente riferito all'opzione grafica, mentre non è possibile dare un giudizio in merito allo sviluppo effettivo usando la prima opzione

- *Profondità*: **7**
  le due opzioni offrono profondità diametralmente opposte: l'uso dell'IDE permette di avere una completa libertà, mentre l'uso dell'opzione grafica è incredibilmente limitante (sebbene sia effettivamente consigliato solo per uno sviluppo di base)

- *Persistenza della memoria*: **6**
  avendo testato attraverso il bot di DialogFlow il giudizio è il medesimo

- *Pricing strategies*: **8**
  piattaforma completamente gratuita e senza vincoli di progettazione, possibilitá di richiedere informazione di prezzo personalizzate a seconda delle interazioni (business type)

- *Piattaforme*: **9**
  comprende pressoché tutte le piattaforme più diffuse

- *Disponibilitá di SDK*: **7**
  l'IDE permette di usare JavaScript e Node.js

Commento personale Il servizio è ottimo ed è utile sia all'utente più navigato sia ad un utente meno esperto. L'integrazione con DialogFlow è assolutamente di centrale importanza, permettendo ad un bot sviluppato con tale servizio di essere implementato senza problemi anche su GupShup.

2.4.5 Microsoft Bot Framework

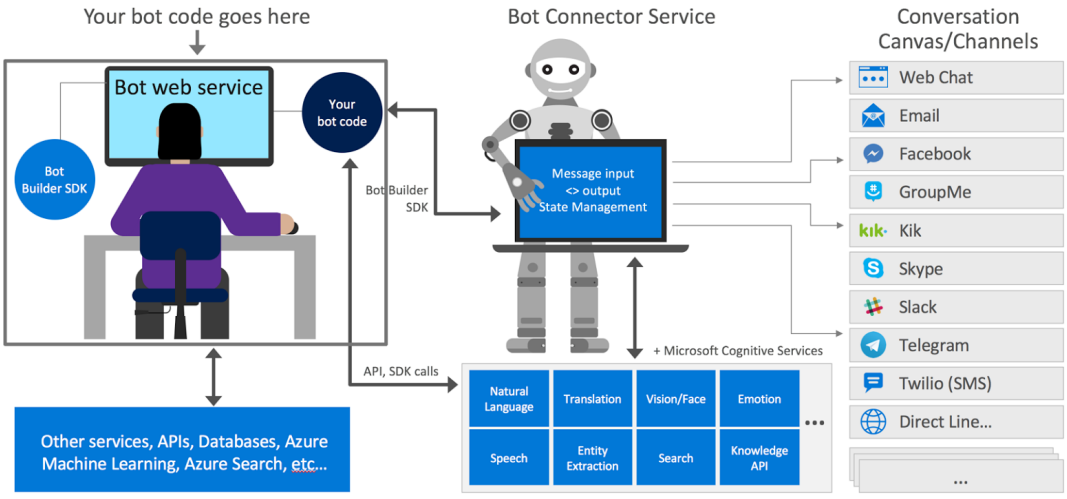
Il frameWork di Windows comprende molti strumenti, ma ci offre un'esperienza troppo macchinosa e non prettamente indirizzata alla progettazione del chatBot come le altre piattaforme. Prima di tutto occorre sottolineare che l'applicazione offre 3 servizi principali:

1. Azure Bot Service - fornisce le risorse necessarie per l'hosting (a pagamento), il deployment e l'editing del bot (tramite editor o interfacciandosi a IDE come ad esempio VisualStudio)

2. Bot Builder - insieme di SDK utili alla costruzione del bot. Nativamente supporta C# e Node.js, mentre si ha la possibilitá di usare tutti gli altri tipi di linguaggi tramite una REST api

3. Bot FrameWork Portal - portale principale di accesso al Framework, permette la manipolazione delle informazioni di base del bot, la creazione di nuovi e prevede una piattaforma di testing

La costruzione di un bot all'interno di Microsoft Bot Framework avviene solo ed esclusivamente tramite programmazione "vecchia maniera", ovvero tramite progettazione su carta e sviluppo con un linguaggio di programmazione a piacere, gestendo le domande e il flow delle risposte a piacimento. Non c'è ovviamente nessun Intent Recognition di base, a meno che non si usino delle chiamate di API NLU (LUIS di Microsoft - a pagamento ovviamente - o esterne). Grazie alla possibilitá di poter sviluppare con righe di codice il proprio bot è ovviamente favorita la personalizzazione e il controllo. È presente inoltre un connector service che collega il codice con qualsiasi tipo di applicazione input/output vogliamo. Sono ovviamente presenti degli strumenti di test e debug indirizzati in particolare a questo argomento. Nota molto positiva è la ricca documentazione, anche perché nella sezione di design del bot è presente la formalizzazione di alcuni concetti molto significativi nello studio dell'efficacia di un bot.



Microsoft Bot FrameWork overview

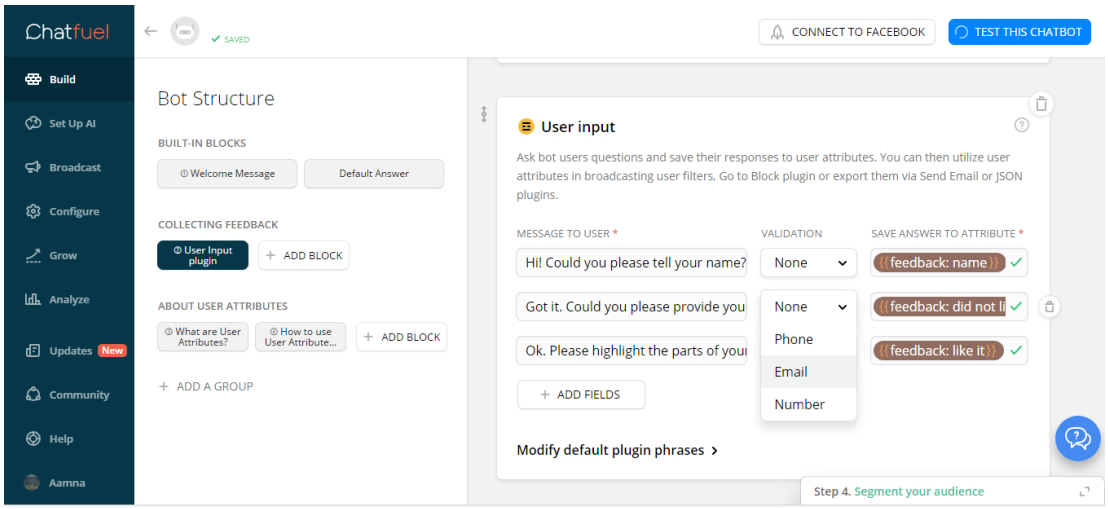
Conclusioni Ci è stato chiesto di "esplorare" il neonato mondo dei bot, in particolare ChatBot alla ricerca della migliore alternativa (in base a parametri di personale importanza) di sviluppo dello stesso. Pur essendo informatici, non avendo paura della possibilitá di tuffarsi sull'apprendimento di un linguaggio di programmazione a noi non noto e apprezzando il poter avere la libertá di scrivere codice, consideriamo questo approccio poco adatto al nostro progetto in quanto ""obsoleto"" e poco flessibile, oltre a richiedere potenzialmente troppo tempo.

2.4.6 Wit.ai

Dopo aver consultato la guida per iniziare e aver cominciato a creare qualcosa, mi son reso conto che mancavano delle funzioni significative. Ho appreso che Facebook sta chiudendo la piattaforma in modo da integrare il "natural language processing (NPL) tools into Messenger Platform 2.1"

2.4.7 ChatFuel

Implementazione Nel servizio è presente solo la possibilitá di maneggiare la struttura del bot solo tramite GUI. Interfaccia molto semplice, presenza di "Blocks" che permettono di modellare l'automatizzazione di azioni (presa input, richieste http, stampa video,..) e "Groups" nella quale racchiudere a piacimento questi blocks. I suddetti blocchi possono essere concatenati in sequenze a seconda del bisogno, ed esistono strutture che permettono di maneggiare queste transizioni (da blocco a blocco - da sequenza a sequenza) in modo quasi intuitivo e completo. Questi blocchi possono avere una vasta gamma di eventi di trigger, ma principalmente vengono richiamati dalla sezione "AI" che permette di inserire delle frasi (con "riconoscimento del contesto") ed eseguire di conseguenza un dato blocco. Dopo aver capito i funzionamenti base della piattaforma dunque è bastato creare un singolo blocco con la giusta presa di input dall'utente e una richiesta http per creare il bot desiderato. Da notare l'impossibilitá di testare il bot su una qualche console ma solo direttamente sulla propria pagina facebook (alla quale si deve per forza collegare il bot). Per completezza del documento, in questa piattaforma è stato necessario ri-adattare il codice in Node.js descritto all'inizio di questo testo, in modo che ricevesse del codice JSON in entrata e uscita leggermente diversi, come scritto sulla repository di github.



Panoramica della piattaforma (user input plugin)

Valutazione

- *Facilitá di progettazione*: **8**
  Esclusa la difficoltá nel trovare il giusto formato JSON, l'intera piattaforma è molto intuitiva (presente solo interfaccia grafica)

- *Tempo di implementazione*: **9**
  2,5 ore, senza contare lo sviluppo del motore su Google Cloud Platform (giá configurato per altri Framework)

- *Profondità*: **6**
  Da un lato è molto limitante (ad esempio, non è possibile fare un parsing di una frase in modo da ottenere specifici parametri, ma si deve dar la possibilitá all'utente di inserirli volta per volta), dall'altro è interessante la possibilitá di usare Plugin modulabili a piacimento (sia interni sia esterni), anche se preimpostati. Presenza di diversi tool di analisi

- *Persistenza della memoria*: **6**
  All'interno di un blocco è possibile tenere traccia delle variabili o parametri dell'utente

- *Pricing strategies*: **8**
  piattaforma gratuita, ma previa creazione di pagina Facebook. Possibilitá di sottoscrivere abbonamento per un servizio migliore

- *Piattaforme*: **3**
  Implementato solo in facebook

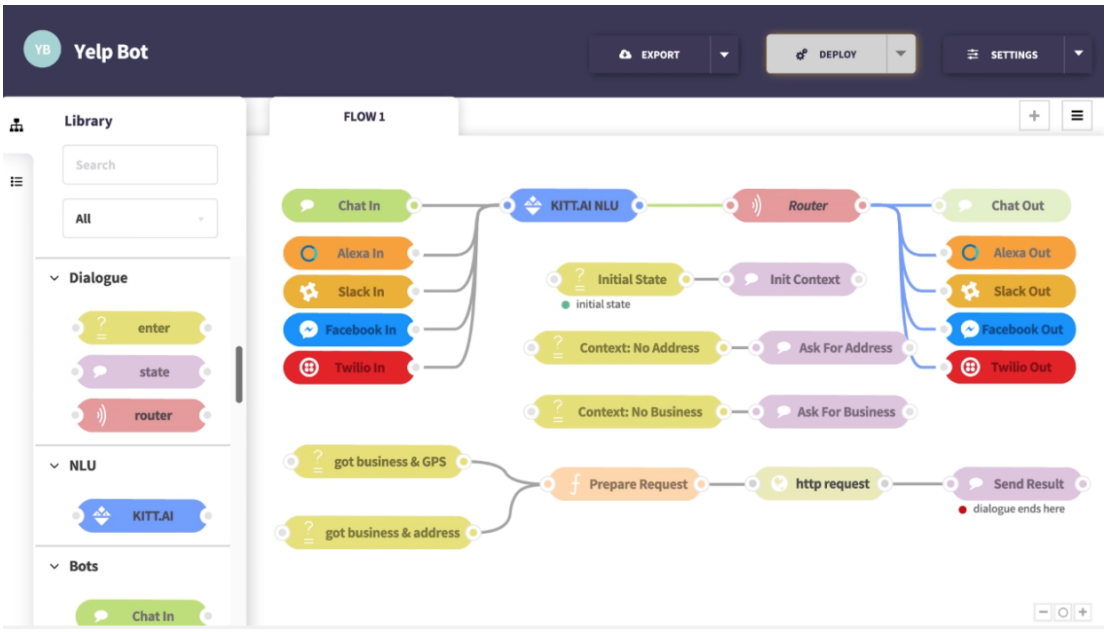
- *Disponibilitá di SDK*: n/a

Commento personale Con questo framework non è possibile creare bot troppo sofisticati, in quanto non sono possibili certe azioni come il riconoscimento automatico di parametri in una frase complessa data dall'utente. È invece, a mio parere, molto consigliato per chi deve gestire una pagina facebook con chat automatica del tipo "botta e risposta". Fornisci i comandi e l'applicazione esegue delle azioni. Stop. Carina la presenza (anche se non richiesta come requisito nelle nostre specifiche) dell'integrazione di vari tool di analisi della crescita e popolaritá, come giustamente mi aspetto da una piattaforma sviluppata unicamente per un social network.

2.4.8 kitt.ai

Implementazione Dopo un'accurata lettura dei docs di riferimento, è stato possibile iniziare a sviluppare il bot di test. Per prima cosa si è dovuto aggiungere alla workZone i nodi di ChatIn e ChatOut (per impostare l'inizio e fine del flow), successivamente in mezzo a essi il nodo router (un piú o meno semplice if condizionato, collegato a ogni enter node (condizioni di entrata del ramo) che permette la scelta del ramo nei vari casi di funzionamento. Questi casi possono essere piú o meno complessi. Per il parsing della frase d'entrata data dall'utente

è stato necessario creare un semplice riconoscimento d'intenti con la piattaforma per NLU (natural language understanding) direttamente da kitt.ai (anche se è possibile collegare diverse NLU, tra cui anche dialogflow). Una volta che sono stati fatti i vari rami di condizione, è bastato aggiungere le varie risposte e un minimo di codice per costruire il Json da passare alla nostra richiesta http. Per questa richiesta è possibile aggiungere un nodo http request che, settando alcuni parametri fondamentali, non ha avuto problemi a consegnarci la risposta che aspettavamo dalla funzione attiva su google cloud. Questa volta è stato necessario modificare leggermente l'output del cloud poiché era scomodo inviare un Json, comodo invece una semplice stringa.



kitt.ai GUI overview

Valutazione

- *Facilitá di progettazione*: **8**
  Dopo una lettura delle docs (piú che altro per capire i vari formati dei messaggi e il funzionamento dei nodi) non è stato assolutamente difficile crearlo. Molto bella e funzionale la GUI, che ti permette di avere una visione chiara del flow del codice

- *Tempo di implementazione*: **8**
  3 ore, la maggior parte speso per varie prove e test

- *Profondità*: **9**
  possibile programmazione in javascript direttamente nel flusso tramite alcuni nodi, integrazione nativa di alcune API e possibilitá di aggiungere

le tue, comodo debug da programmare come vuoi e vasta gamma di nodi funzione per poter creare quello che vuoi

- *Persistenza della memoria*: **10**
  Ho trovato molto semplice e comodo "giocare" con le variabili globali o locali (del messaggio ricevuto). La docs è molto dettagliata su come usare e creare le varie entitá

- *Pricing strategies*: **8**
  Il framework è in fase beta, ma è possibile richiedere e ricevere una key apposta. Gratis fino a un certo numero di interrogazioni, poi richiesta il passaggio a un piano a pagamento

- *Piattaforme*: **10**
  la maggior parte delle piattaforme Business e Private sono supportate

- *Disponibilitá di SDK*: n/a

Commento personale Complessivamente questo Framework sembra un buon candidato alla fase finale del nostro progetto, poiché concilia facilitá di utilizzo con possibilitá abbastanza profonda di personalizzazione. Sarebbe necessario verificare il comportamento del Framework con progetti più ampi. Purtroppo è ancora in fase beta, quindi il suo futuro è abbastanza incerto, ma è stato comunque interessante poter apprendere una logica diversa su cui è stato possibile approfondire il concetto di ChatBot.

2.5 Commenti Finali e selezione del framework

Il framework che più ci ha convinti é **DialogFlow**, che unisce alla semplicitá e intuitivitá di sviluppo del bot una grande profonditá di azioni che é possibile compiere, senza rinunciare a svariati tool di apprendimento e gestione che a noi sono risultati comodi. Note di merito vanno comunque a PandoraBots, che permette di costruire il proprio riconoscimento d'intenti, e kitt.ai, che utilizza la migliore interfaccia tra tutti i sopracitati ambienti.

3 Caso di studio: PolimiHelp

3.1 Architettura

Nel corso delle ricerche del progetto, in particolare durante la fase di scelta del framework, si é cercato di formalizzare l'architettura di un generico chatbot. Il risultato é proposto in **appendice A**, alla quale questa sezione fa riferimento. Si possono vedere 4 parti principali, collegate nelle modalitá spiegate nello schema:

1. **Servizi di Input/Output**: come dice il nome, l'applicazione che ci permette di raccogliere l'input e mostrare l'output all'utente.

2. **Framework**: insieme degli strumenti che il framework (nel nostro caso dialogFlow) ci consente di utilizzare, dal riconoscimento del linguaggio (NLU) all'interfaccia di progettazione della conversazione. Puó essere usata per "chiamare" API esterne o per fare richieste web qualsiasi (http/..). A seconda di come si é progettato il bot questa sezione puó essere opzionale.

3. **Server**: il framework puó appoggiarsi a un codice attivo su un server per svolgere alcune funzioni e richiedere informazioni. É da sottolineare il fatto che se da una parte é necessario, una volta cambiata piattaforma di lavoro, riprogrammare il framework, il codice attivo sul server (a meno di piccoli dettagli) rimane invariato.

4. **Raccolta informazioni**: prese da API esterne o scraping.

5. **Mondo esterno**: servizi e pagine presenti in rete.

3.2 Scopo

La definizione delle caratteristiche funzionali inizia con il dividerle in utenti che effettuano la propria registrazione con la coppia <Codice persona, Password> e in utenti che non lo fanno. La registrazione non é ovviamente obbligatoria, ma gli utenti loggati avranno a disposizione piú interazioni personalizzate col bot, oltre che avere tutte quelle possibili per gli utenti normali non loggati.

UnLogged Users

1. *Orario universitario*: richiesta orario delle lezioni, previo inserimento di tutti quei parametri che servono per identificare il corso (nome, corso di studio, sede).

2. *Aule libere*: richiesta elenco aule libere, previa scelta del giorno e orario.

3. *Orario aula*: richiesta elenco corsi e orari di una specifica aula.

4. *Posizione aula*: restituisce semplicemente l'edificio in cui é l'aula (utile per le matricole).

5. *Calendario accademico*: richiesta orario accademico, manda il PDF presente sul sito del Politecnico.

Logged Users

1. *Orario universitario personalizzato*: stessa cosa della funzionalitá parte utente loggato, ma con i dati a disposizione é possibile avere maggiore accuratezza e non é richiesto l'inserimento dei dati.

2. *Esami*: richiesta elenco degli esami a cui ci si é iscritti.

3. *Informazione esami*: informazioni su esami a cui si é iscritti (i.e. aula, orario).

4. *Carriera didattica*: restituzione informazioni generali della propria carriera (e.g. media, crediti).

5. *Avvisi generici*: possibilitá di settare avvisi generici con qualsiasi evento di trigger (inizialmente pensata solo uscita voti esami con relativa votazione).

3.3 Modalitá di interazione con gli utenti

Inizialmente si pensava di supporre che il bot potesse rispondere a un set di comandi predefiniti e dati da noi, non lasciando spazio all'utente per eventuali frasi complesse da interpretare. Questo approccio ci avrebbe permesso di tralasciare quasi completamente la parte sul parsing e riconoscimento degli intenti. Man mano che cominciavamo a sviluppare il nostro progetto peró questo approccio ci é sembrato limitante, poiché la vera natura del nostro progetto era di "esplorazione" di questi nuovi ambienti di sviluppo. Abbiamo quindi convenuto che la parte interessante sarebbe stata appunto confrontare le varie soluzioni giá presenti di NLU (natural language understanding) e rendere la conversazione col bot il piú vicino possibile a quella con un umano. Ecco quindi che si delinea l'importanza dell'utilizzo di un Framework che abbia queste possibilitá.

3.4 Requisiti non funzionali

Sotto questa voce vanno i requisiti e le caratteristiche che impongono vincoli allo sviluppo del bot, sotto diversi punti di vista. Per ogni problema si cercherá quindi una possibile soluzione di implementazione, che non andremo a sviluppare nella demo in quanto non riteniamo necessario in questo progetto approfondire ulteriormente.

3.4.1 Prestazioni e Price strategies

Secondo il modello posto in appendice A (architettura), un bot in generale si puó rappresentare con due parti distinte, poste totalmente o in parte su un server esterno. La scelta del server influenzerá sicuramente il numero di interrogazioni che si possono fare, la velocitá di risposta e l'usabilitá in generale, a discapito del prezzo che la piattaforma di hosting ci fará pagare per avere questi servigi. Se si utilizza una soluzione "locale" in cui il proprio pc viene utilizzato come server si avranno sicuramente degli svantaggi rispetto a una soluzione cloud, ma il prezzo sará nullo e le limitazioni (se non date da hardware) non esisterebbero. Ad ora, esplorando le varie possibili scelte, per la fase di testing della demo é stato possibile sempre usufruire delle opzioni "free" della piattaforma esaminata. Questo include sia parte di interazione utente che Hosting Server. Inoltre, rientra in questa sezione anche la possibilitá di ridurre query rindondanti attraverso l'implementazione di un minimo di *caching*. Non é una parte da sottovalutare in quanto potrebbe risparmiarci un parte consistente di interrogazioni inutili al

sistema e migliorare l'efficienza generale. Si pensava quindi di usare inizialmente un unico file di materializzazione dei dati, che viene "guardato" dal codice ogni volta che giunge una nuova interrogazione, e aggiornato se i dati richiesti non sono presenti o troppo vecchi per essere attendibili. É un'implementazione semplice di caching, ma per i nostri scopi puó funzionare.

3.4.2 Privacy e persistenza

Oltre alle informazioni pubbliche dell'utente, dobbiamo trattare dati personali (e.g. password, codice persona ...) e riservati, che nè gli sviluppatori nè soggetti esterni dovrebbe essere in grado di leggere. Queste informazioni doverebbero, se non vogliamo che l'utente debba sempre inserire le sue credenziali, essere mantenute in qualche database esterno criptato in modo che neanche noi possiamo accedere al contenuto se non per interrogazione da parte del codice. Purtroppo questa problematica é una parte complessa e non strettamente rilevante per il progetto. Non analizzeremo dunque piú a fondo questa sezione.

3.4.3 Data extraction

Il nostro progetto é strettamente correlato alle informazioni esterne che dobbiamo cercare all'interno (nel nostro caso) del sito del politecnico/beep, che purtroppo non é fornita di comode API che possiamo chiamare nel codice. Viene quindi visto indispensabile un cosiddetto *motore di scraping*, ovvero una tecnica informatica di estrazione di dati da un sito web per mezzo di programmi software. Anch'esso é un argomento molto (forse troppo) vasto per parlarne in queste poche righe, e non strettamente rilevante per lo scopo del progetto. Ci siamo visti purtroppo costretti a non implementarlo nella prima demo del bot. Al suo posto abbiamo pensato di utilizzare un motore "fake" costruito su un foglio di testo, ma di questo se ne parlerá nella sezione dedicata allo sviluppo della demo. Abbiamo ovviamente fatto ricerche, seppur quasi superficiali, su questo argomento. É venuto fuori che esistono decine e decine soluzioni (free o a pagamento) che offrono web scraping di varia profonditá, integrazioni con API e servizi. Uno in particolare ha attirato la nostra attenzione: un portale free che offre soluzioni di scraping *in teoria facile da programmare* che risponde al nome di **Portia**. Qualche prova é stata fatta per vedere che cosa si poteva ottenere e per adattare al meglio la nostra demo, in una previsione di implementazione futura. Si é concluso che il Json é sempre il mezzo piú semplice e veloce per simulare i dati estratti da questi motori.

4 Flusso di funzionamento

Vengono sotto proposti due tipi differenti di rappresentazioni grafiche, che nel loro insieme dovrebbero comporre la parte dedicata all'interazione con l'utente.

La prima (presentata in **Appendice B**) infatti, simile a un diagramma di flusso, mette ordine sul percorso che la conversazione deve avere, presentando il

tipo di risposte che l'utente si dovrebbe aspettare dopo determinate interrogazioni.

La seconda (presentata in **Appendice C**), é invece piú simile a un diagramma delle classi e cerca di mettere in chiaro relazioni e caratteristiche degli argomenti della conversazioni (chiamati **contesti** o context). Ogni volta che un utente interagisce col bot, a seconda del tipo di input inserito viene riconosciuto e attivato un preciso **intento** (o intent) che svolgerá le funzioni programmate e risponderá nella maniera programmata. L'insieme di piú intenti forma un contesto. Proprio quest'ultimo schema, essendo piú elaborato, ha bisogno di essere spiegato con una legenda (vedi Fig. 1) e attraverso alcune regole base che aiutano a leggere l'appendice:

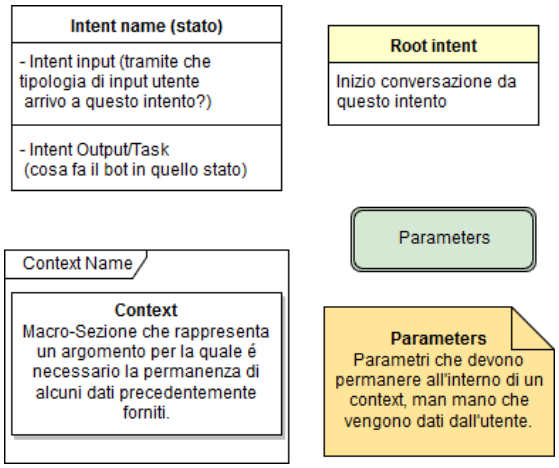


Figura 1: Legenda appendice B

- Da un contesto posso passare a qualsiasi altro contesto o restare al suo interno a seconda dell'input fornito (e quindi dell'intento riconosciuto).
- Una conversazione all'interno di un contesto puó cominciare da qualsiasi "root intent" (vedi legenda) dello stesso contesto.
- Il passaggio tra due contesti può avvenire in qualsiasi momento: a seconda dell'intento che ne permette il passaggio, i parametri vengono tenuti o meno.
- Non possono esistere più "root intent" con le stesse frasi di trigger, possono invece esistere degli intent normali con le stesse frasi.
- La relazione padre/figlio (freccia vuota) esiste quando un intento padre contiene nell'input piú parametri riconosciuti. Per la scelta dell'intento che deve fornire l'output in questa relazione si va a prendere il figlio che necessita di meno parametri per essere "attivato" (seguendo la gerarchia grafica).
- Gli intenti tra gli asterischi sono quelli che possono essere usati per le risposte di intenti che partono dallo stesso contesto. Un intento padre passa questa proprietá ai figli.

5 Costruzione di PolimiHelp

Dopo aver quindi svolto un lungo lavoro di ricerca abbiamo le conoscenze necessarie per costruire la demo del bot. Appunto perchè si tratta di una prova

optiamo per una versione "limitata" del progetto, che però ci permette di utilizzare e consolidare le conoscenze acquisite nella parte precedente. Tutte le limitazioni rispetto all'idea originale sono riportate nella sezione 5.3, mentre nelle prime due sono descritti in linea di massima le principali componenti e le nostre soluzioni di implementazione. Avendo ben definito queste due parti, ci è venuto naturale dividerci i compiti tra chi si sarebbe occupato di scrivere il codice nella parte server (Stefano) e chi si sarebbe invece dedicato alla struttura della conversazione (Paolo).

5.1 ServerSide

A livello server abbiamo ritenuto opportuno seguire lo stesso procedimento adottato per il test delle piattaforme, ovvero impostare un webhook. A differenza del test con l'API di `https://developer.worldweatheronline.com` abbiamo dovuto scrivere del codice Javascript adatto a gestire le richieste del nostro PolimiBot. Per impostare il webhook per tale scopo abbiamo deciso di utilizzare i servizi offerti dalla piattaforma Google Cloud Platform e utilizzare una Cloud Function con il codice Javascript sopra citato. Il principio di base è eseguire una specifica funzione a seconda della richiesta dell'utente: più nello specifico abbiamo deciso di utilizzare uno switch basato sul contesto inviato all'interno del JSON del richiedente. Inizialmente si era pensato di usare uno switch sugli intenti: tale approccio è risultato non adatto poiché non permetteva di gestire multiple richieste consecutive nel medesimo contesto. (es. chiedere la posizione di più aule consecutivamente) La gestione dei dati degli eventuali utenti è stata una scelta più difficile: dapprima abbiamo provato ad utilizzare il servizio di Cloud SQL (sempre offerto da Google Cloud Platform), ma non avendo dimestichezza con i database abbiamo deciso di non seguire questa idea. Abbiamo quindi ragionato sui corsi di studio e sulle competenze acquisite e abbiamo deciso di utilizzare dei file JSON come già praticato durante il corso di Ingegneria del Software: oltre alla motivazione sopra descritta è opportuno sottolineare che il frontend comunica con il backend utilizzando dei JSON, quindi ci è sembrato naturale scegliere quel formato. L'uso dei file JSON ci ha permesso non solo di gestirle multiple sorgenti di dati, ma anche di avere librerie già adibite alla lettura e alla manipolazione dei file JSON: tali file possono inoltre essere scorporati dal codice e usati in un server locale, qualora ce ne fosse la necessità.

5.2 UX (DialogFlow)

La parte in cui viene modellata la conversazione macchina-utente è interamente contenuta in DialogFlow. In buona parte la piattaforma è descritta nella sezione dedicata alla scelta del framework. Si è cercato di seguire il più fedelmente possibile lo schema descritto sopra di interazione tra i contesti. Usando DialogFlow non ho avuto molta difficoltà a creare le relazioni tra intenti che cercavamo, più difficile invece è stato creare i contesti e utilizzarli in modo efficace (con il passaggio di parametri sopra descritto nel passaggio tra contesti). Il riconoscimento di parametri è molto valido se si tratta di parametri di sistema (e.g. data, ora,

period), mentre è stato necessario crearne di personali per aule e corsi di studi. Questa ultima parte è abbastanza noiosa in quanto, per avere la maggior accuratezza possibile, dovresti inserire tutti i possibili valori con i relativi sinonimi (a meno che non voglia rischiare di riconoscere parametri non corretti). Tutto sommato però, dopo aver passato qualche ora a ragionare sulle diverse relazioni tra contesti e intenti, è venuto fuori un bot completo, come appunto descritto nello schema a oggetti.

5.3 Limitazioni

Ecco i principali punti che descrivono in cosa questa demo si discosta dal progetto completo:

- Motore di Scraping costruito "ad-Hoc" (informazioni non dinamiche), cioè posto come formato json come costante all'interno del codice sul server.
- Implementata solo parte utenti NON loggati.
- Persistenza assente.
- Non completata la parte server di orari lezioni, implementata solo su dialogflow.

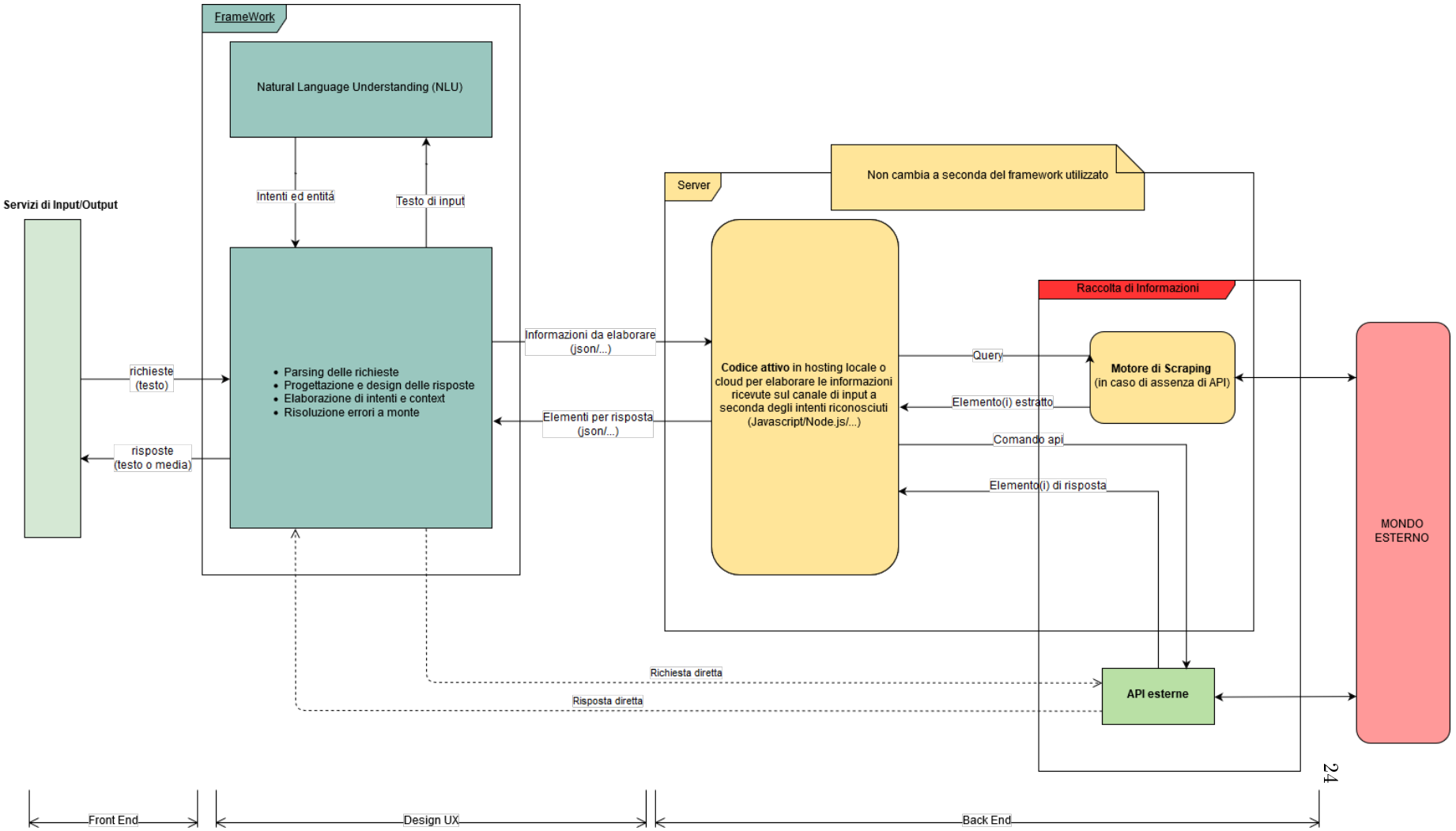
6 Sviluppi Futuri

Cosa ci aspetta il futuro? Ecco cosa ci piacerebbe implementare e come intenderemmo farlo:

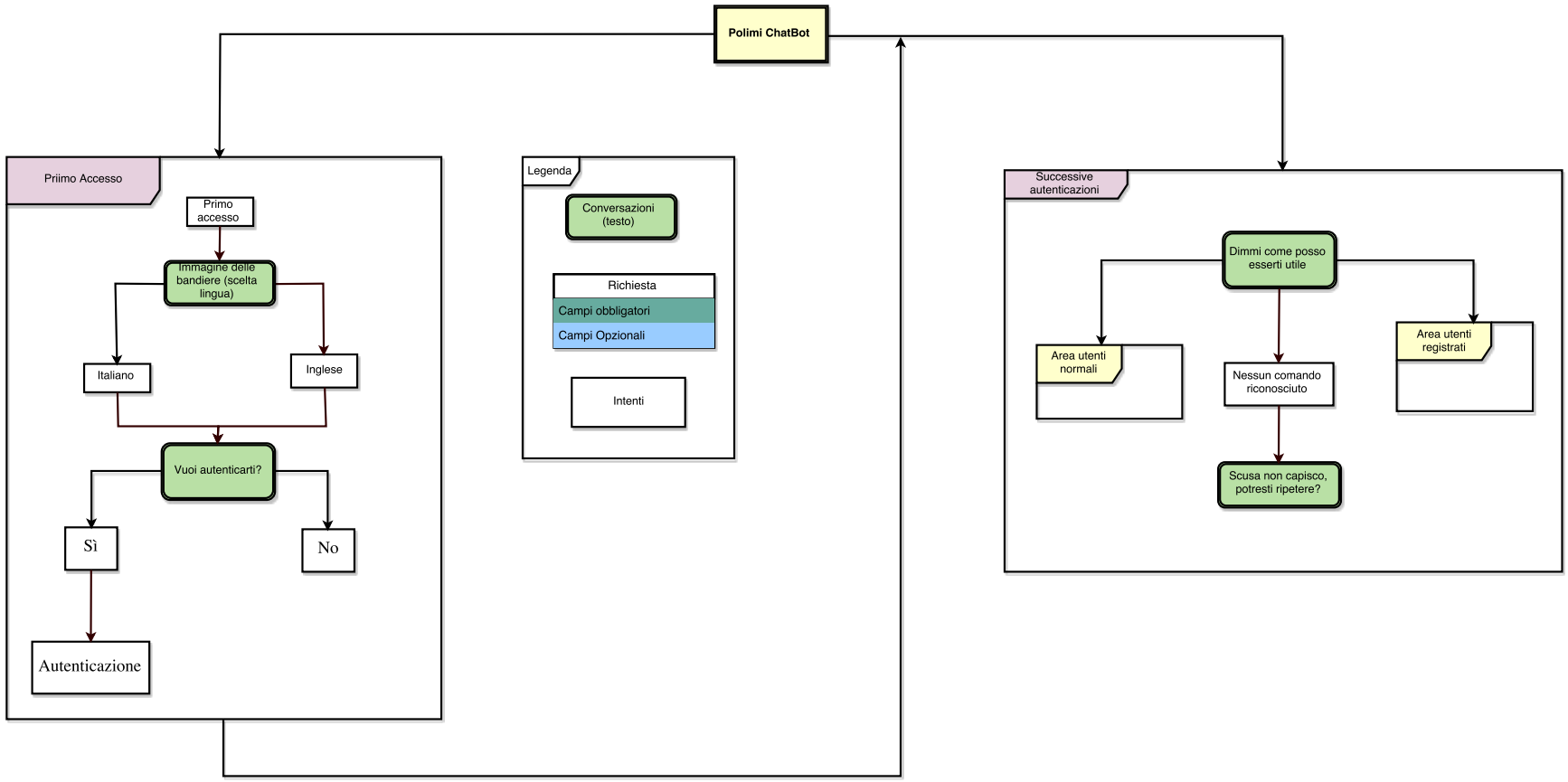
- Completare la parte degli utenti non loggati con tutte le sue parti funzionanti nel codice server.
- Utilizzare uno scraping che permetta di manipolare informazioni dinamiche: può essere comunque utilizzato un motore che "crei" un json in modo da non dover cambiare la struttura del nostro codice, o perlomeno apportare solo piccole modifiche.
- Trovare la migliore soluzione per quanto riguarda la memorizzazione dei dati. Si pensava inizialmente di mettere tutto dentro Database con chiave univoca il codice che rappresenta l'utente (preso dalla piattaforma di chat che sta usando).
- Implementare il riconoscimento utente tramite controllo codice univoco della piattaforma (dialogflow in questo caso). A seconda delle informazioni che troviamo l'esperienza verrà personalizzata.
- Cercare soluzioni per ottenere un livello di pivacy accettabile in modo da poter richiedere informazioni delicate quali codice persona e password.

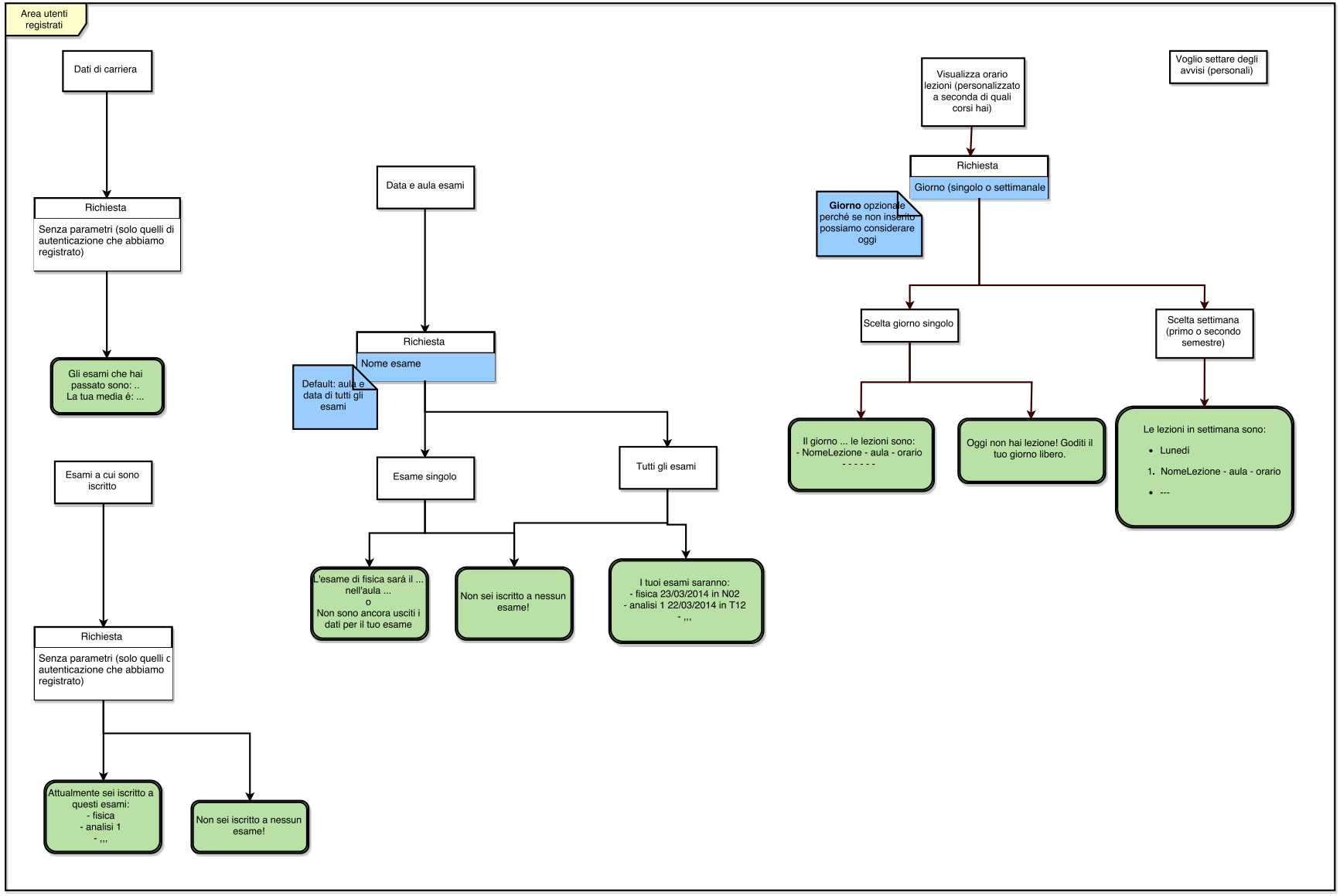
- Trovare un modo efficace per utilizzare queste informazioni tramite il motore di scraping sopra citato. Ovviamente devono essere utilizzate in modo da mantenere il livello di privacy minimo richiesto.
- Implementare quindi tutte le funzionalitá utente loggato, meno gli avvisi personalizzati.
- Costruire un evento trigger per l'invio delle notifiche, parte che richiede uno studio piú approfondito.

A Architettura di un chatbot

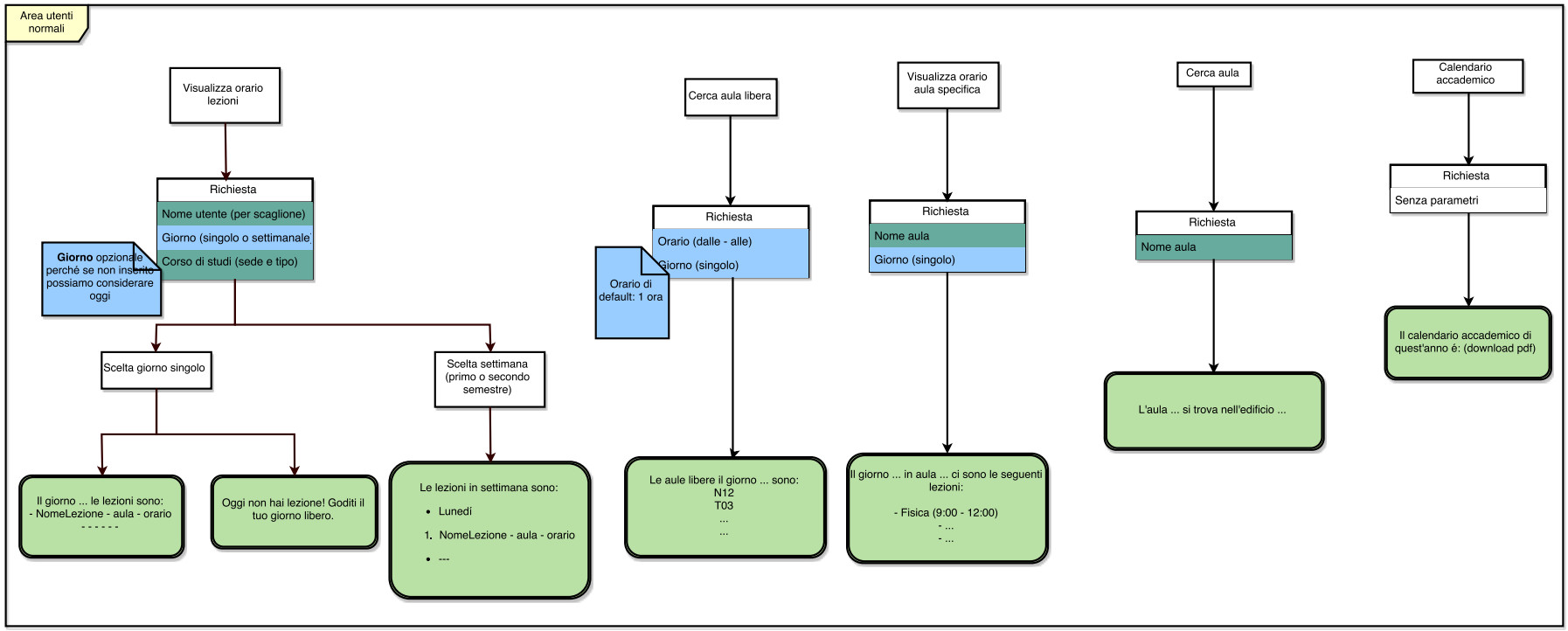

FrameWork
Natural Language Understanding (NLU)
Intenti ed entitá
Testo di input
Servizi di Input/Output
richieste
(testo)
risposte
(testo o media)
• Parsing delle richieste
• Progettazione e design delle risposte
• Elaborazione di intenti e context
• Risoluzione errori a monte
Informazioni da elaborare
(json/...)
Elementi per risposta
(json/...)
Server
Non cambia a seconda del framework utilizzato
Codice attivo in hosting locale o cloud per elaborare le informazioni ricevute sul canale di input a seconda degli intenti riconosciuti (Javascript/Node.js/...)
Raccolta di Informazioni
Query
Elemento(i) estratto
Motore di Scraping
(in caso di assenza di API)
Comando api
Elemento(i) di risposta
MONDO
ESTERNO
Richiesta diretta
Risposta diretta
API esterne
Front End
Design UX
Back End

B Schema della conversazione


Polimi ChatBot
Priimo Accesso
Primo accesso
Immagine delle bandiere (scelta lingua)
Italiano
Inglese
Vuoi autenticarti?
Sì
No
Autenticazione
Legenda
Conversazioni (testo)
Richiesta
Campi obbligatori
Campi Opzionali
Intenti
Successive autenticazioni
Dimmi come posso esserti utile
Area utenti normali
Nessun comando riconosciuto
Area utenti registrati
Scusa non capisco, potresti ripetere?

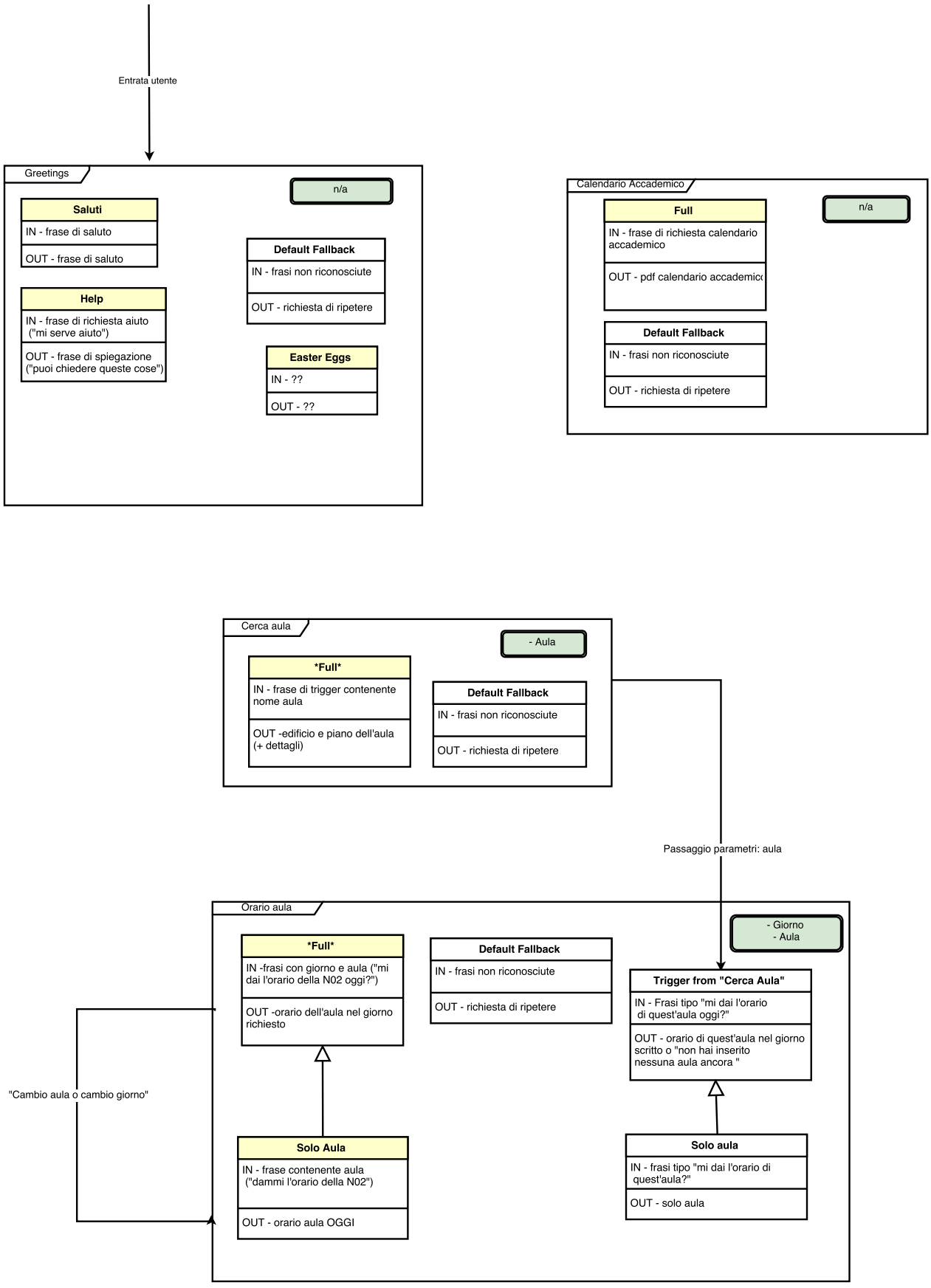

Area utenti registrati
Dati di carriera
Richiesta
Senza parametri (solo quelli di autenticazione che abbiamo registrato)
Gli esami che hai passato sono: ..
La tua media é: ...
Esami a cui sono iscritto
Richiesta
Senza parametri (solo quelli c autenticazione che abbiamo registrato)
Attualmente sei iscritto a questi esami:
- fisica
- analisi 1
- ,,,
Non sei iscritto a nessun esame!
Data e aula esami
Richiesta
Nome esame
Default: aula e data di tutti gli esami
Esame singolo
Tutti gli esami
L'esame di fisica sará il ... nell'aula ...
o
Non sono ancora usciti i dati per il tuo esame
Non sei iscritto a nessun esame!
I tuoi esami saranno:
- fisica 23/03/2014 in N02
- analisi 1 22/03/2014 in T12
- ,,,
Visualizza orario lezioni (personalizzato a seconda di quali corsi hai)
Richiesta
Giorno (singolo o settimanale
Giorno opzionale perché se non inserito possiamo considerare oggi
Scelta giorno singolo
Scelta settimana (primo o secondo semestre)
Il giorno ... le lezioni sono:
- NomeLezione - aula - orario
- - - - - -
Oggi non hai lezione! Goditi il tuo giorno libero.
Le lezioni in settimana sono:
Lunedì
NomeLezione - aula - orario

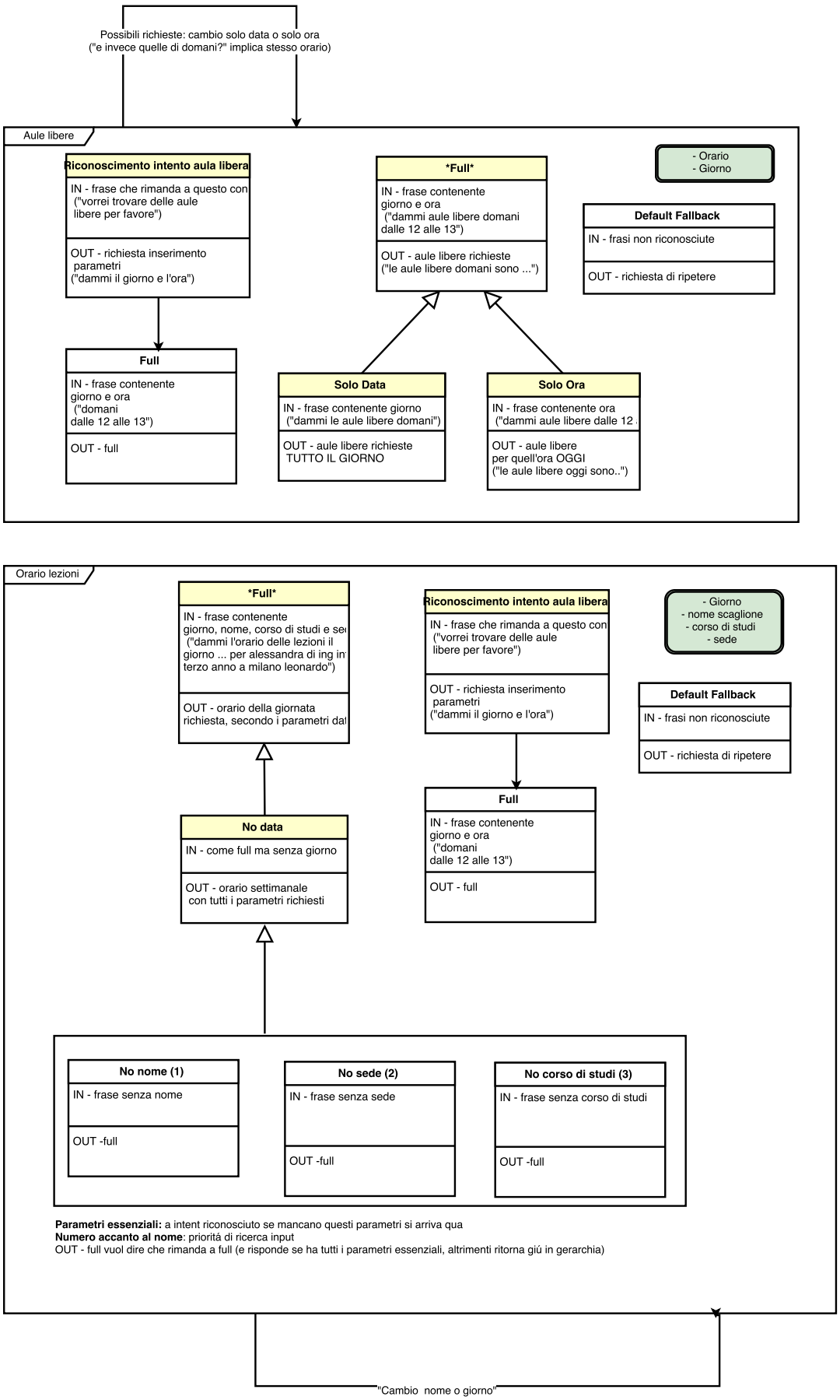
Voglio settare degli avvisi (personali)


Area utenti normali
Visualizza orario lezioni
Richiesta
Nome utente (per scaglione)
Giorno (singolo o settimanale)
Corso di studi (sede e tipo)
Giorno opzionale perché se non inserito possiamo considerare oggi
Scelta giorno singolo
Scelta settimana (primo o secondo semestre)
Il giorno ... le lezioni sono:
- NomeLezione - aula - orario
- - - - - -
Oggi non hai lezione! Goditi il tuo giorno libero.
Le lezioni in settimana sono:
Lunedì
NomeLezione - aula - orario

Cerca aula libera
Richiesta
Orario (dalle - alle)
Giorno (singolo)
Orario di default: 1 ora
Le aule libere il giorno ... sono:
N12
T03
...
...
Visualizza orario aula specifica
Richiesta
Nome aula
Giorno (singolo)
Il giorno ... in aula ... ci sono le seguenti lezioni:
- Fisica (9:00 - 12:00)
- ...
- ...
Cerca aula
Richiesta
Nome aula
L'aula ... si trova nell'edificio ...
Calendario accademico
Richiesta
Senza parametri
Il calendario accademico di quest'anno è: (download pdf)

C Diagramma dei contesti


Entrata utente
Greetings
n/a
Saluti
IN - frase di saluto
OUT - frase di saluto
Default Fallback
IN - frasi non riconosciute
OUT - richiesta di ripetere
Help
IN - frase di richiesta aiuto ("mi serve aiuto")
OUT - frase di spiegazione ("puoi chiedere queste cose")
Easter Eggs
IN - ??
OUT - ??
Calendario Accademico
n/a
Full
IN - frase di richiesta calendario accademico
OUT - pdf calendario accademic(
Default Fallback
IN - frasi non riconosciute
OUT - richiesta di ripetere
Cerca aula
- Aula
Full
IN - frase di trigger contenente nome aula
OUT -edificio e piano dell'aula (+ dettagli)
Default Fallback
IN - frasi non riconosciute
OUT - richiesta di ripetere
Passaggio parametri: aula
Orario aula
- Giorno
- Aula
Full
IN -frasi con giorno e aula ("mi dai l'orario della N02 oggi?")
OUT -orario dell'aula nel giorno richiesto
Default Fallback
IN - frasi non riconosciute
OUT - richiesta di ripetere
Trigger from "Cerca Aula"
IN - Frasi tipo "mi dai l'orario di quest'aula oggi?"
OUT - orario di quest'aula nel giorno scritto o "non hai inserito nessuna aula ancora "
"Cambio aula o cambio giorno"
Solo Aula
IN - frase contenente aula ("dammi l'orario della N02")
OUT - orario aula OGGI
Solo aula
IN - frasi tipo "mi dai l'orario di quest'aula?"
OUT - solo aula


Possibili richieste: cambio solo data o solo ora
("e invece quelle di domani?" implica stesso orario)
Aule libere
Riconoscimento intento aula libera
IN - frase che rimanda a questo con
("vorrei trovare delle aule
libere per favore")
OUT - richiesta inserimento
parametri
("dammi il giorno e l'ora")
Full
IN - frase contenente
giorno e ora
("dammi aule libere domani
dalle 12 alle 13")
OUT - aule libere richieste
("le aule libere domani sono ...")
- Orario
- Giorno
Default Fallback
IN - frasi non riconosciute
OUT - richiesta di ripetere
Full
IN - frase contenente
giorno e ora
("domani
dalle 12 alle 13")
OUT - full
Solo Data
IN - frase contenente giorno
("dammi le aule libere domani")
OUT - aule libere richieste
TUTTO IL GIORNO
Solo Ora
IN - frase contenente ora
("dammi aule libere dalle 12
OUT - aule libere
per quell'ora OGGI
("le aule libere oggi sono..")
Orario lezioni
Full
IN - frase contenente
giorno, nome, corso di studi e se
("dammi l'orario delle lezioni il
giorno ... per alessandra di ing in
terzo anno a milano leonardo")
OUT - orario della giornata
richiesta, secondo i parametri dat
Riconoscimento intento aula libera
IN - frase che rimanda a questo con
("vorrei trovare delle aule
libere per favore")
OUT - richiesta inserimento
parametri
("dammi il giorno e l'ora")
- Giorno
- nome scaglione
- corso di studi
- sede
Default Fallback
IN - frasi non riconosciute
OUT - richiesta di ripetere
No data
IN - come full ma senza giorno
OUT - orario settimanale
con tutti i parametri richiesti
Full
IN - frase contenente
giorno e ora
("domani
dalle 12 alle 13")
OUT - full
No nome (1)
IN - frase senza nome
OUT -full
No sede (2)
IN - frase senza sede
OUT -full
No corso di studi (3)
IN - frase senza corso di studi
OUT -full
Parametri essenziali: a intent riconosciuto se mancano questi parametri si arriva qua
Numero accanto al nome: priorità di ricerca input
OUT - full vuol dire che rimanda a full (e risponde se ha tutti i parametri essenziali, altrimenti ritorna giú in gerarchia)
"Cambio nome o giorno"

D Tabella framework

IBM Watson Conversation Service	**AgentBot**
Twyla	**Pypestream**
Live Agent	**Digital Genius**
Semantic Machines	**Msg.ai**
wit.ai	**rasa NLU**
DialogFlow	**Microsoft Bot Framework**
LUIS	**Chatfuel**
Pandorabots	**ChatterBot**
Octane.ai	**Rebot.me**
ManyChat	**FlowXO**
Gubshup	**reply.ai**
kitt.ai	**it's alive**
Chatscript	

References & Documents

[1] Repository GitHub
`https://github.com/Roncax/PolimiChatBot`

[2] Utile articolo di design principles
`https://docs.microsoft.com/en-us/bot-framework/bot-service-design-principles`

[3] Google DialogFlow docs
`https://dialogflow.com/docs/getting-started/basics`

[4] Altro utile articolo sul design di un chatbot
`https://chatbotsmagazine.com/design-framework-for-chatbots-aa27060c4ea3`

[5] Elenco completo bot presi in considerazione
`https://docs.google.com/spreadsheets/d/1qa4G0vpVzjYXBoZV4802b5BbwxpEahAN3QkJ2494jy0/edit#gid=0`