Bot Telegram per Ricerca Lezioni Online

Anno Accademico: 2020/2021
Gabriele Panciotti 336131
Ingegneria del Software, Alfredo Milani

Abstract

Il progetto consiste nella progettazione e l'implementazione di un bot Telegram, con cui qualsiasi studente o professore dell'Universita di Perugia, potrà trovare il link dell'aula Teams per la lezione cercata.
La funzionalità in più rispetto alla pagina web è la possibilità di impostare il proprio corso di appartanenza e l'anno di corso e ricevere in automatico il link a tutte le lezioni da seguire durante l'anno.

Obiettivo

L'obiettivo del progetto è quello di realizzare un bot telegram che semplifichi a studenti e professori, la ricerca dell'aula per le lezioni svolte a distanza tramite la piattaforma Teams.

Analisi dei Requisiti

Requisiti Funzionali

- Il sistema permettera una **ricerca** tra tutte le lezioni proposte dall'Università di Perugia, svolte in modalità a distanza
- Il sistema permettera una ricerca della lezione desiderata tramite un semplice messaggio di testo contente il nome del corso
- Il sistema permettera una ricerca della lezione desiderata tramite un messaggio di testo contente il docente che svolge l'insegnamento
- Il sistema restituirà un messaggio per ogni risultato trovato dalla ricerca, questo sarà composto da:
    - Nome della Lezione
    - Docente
    - Corso di Laurea
    - Link all'aula Teams
- L'utente avrà la possibilità di impostare un **filtro di ricerca** sul Corso di Laurea, in modo da filtrare i risultati a solo quelli appartenti al Corso di Laurea scelto, il filtrò rimarrà impostato per tutte le future ricerche, fino alla rimozione da parte dell'utente del filtro precedentemente impostato
- Il sistema permetterà di impostare un proprio **profilo** selezionando il proprio dipartimento, il corso di laurea, il tipo di corso e l'anno di corso
- Il sistema permetterà all'utente, una volta impostato il proprio profilo, di ricevere un messaggio contenente l'**elenco di tutte le lezioni** del corso e dell'anno scelto, con il relativo link di ogni lezione

Requisiti Non Funzionali

Requisiti di **Prodotto**:

- Il sistema dovrà essere **disponibile** a qualsiasi utente in possesso di un account Telegram, interessato alla visione di una lezione offerta dall'Università di Perugia, questo comprende studenti iscritti all'Università, professori ed eventuali ospiti
- Il sistema dovrà **garantire la fruizione** del servizio tutti i giorni a qualsiasi orario, questo tramite una VPS su cui verrà caricato il bot (non ancora implementato)
- Il sistema dovrà garantire adeguate **prestazioni** per lo svolgimento dei servizi
- Il sistema dovrà garantire un'adeguata **facilità** d'uso da parte dell'utente, sfruttando un interfaccia e dei comandi con bottoni semplici e facili da utilizzare
- Il sistema dovrà garantire un adeguata **affidabilità**, mantenendo sempre aggiornati i link delle lezioni e garantendo l**'integrità** della chat contenente le ricerche effettuate
- Il sistema dovrà garantire un adeguata **privacy** dell'utente, per farlo ho sviluppato il codice in modo da non salvare alcun dato degli utenti in server esterni, ma esclusivamente nel dispositivo dell'utente e nei server di Telegram

Requisiti **Organizzativi**:

- L'utente per poter utilizzare i servizi offerti deve essere in possesso di un numero cellulare e di un account Telegram associato a tale numero
- L'utente per poter utilizzare i servizi offerti deve essere in possesso di un qualsiasi tipo di dispositivo su cui è possibile scaricare Telegram (Android, IOS, Windows, Linux, macOs…)

Requisiti **Esterni**:

- Il sistema non deve rilasciare ai suoi operatori nessuna informazione personale relativa ai clienti
- I dati trattati dal sistema devono rispettare tutte le norme previste dal Regolamento Europeo in materia di protezione dei dati personali

Glossario:

- Bot: Programmi automatizzati che, su richiesta dell'utente, svolgono compiti ben precisi e spesso molto utili per trovare informazioni o determinati tipi di file
- VPS: acronimo di Virtual Private Server ed è una tipologia di hosting che prevede che un server fisico venga suddiviso virtualmente (tramite un hypervisor) in più "ambienti privati", quindi dedicati interamente ad un singolo utente/azienda, che può utilizzarne a pieno le risorse di quell'ambiente (RAM, CPU e spazio di archiviazione)
- API:(acronimo di Application Programming Interface, ovvero Interfaccia di programmazione delle applicazioni) sono set di definizioni e protocolli con i quali vengono realizzati e integrati software applicativi. Consentono ai tuoi prodotti o servizi di comunicare con altri prodotti o servizi senza sapere come vengono implementati, semplificando così lo sviluppo delle app e consentendo un netto risparmio di tempo e denaro
- Web Scraping: Tecnica informatica di estrazione di dati da un sito web per mezzo di programmi software. Di solito, tali programmi simulano la navigazione umana nel World Wide Web utilizzando l'Hypertext Transfer Protocol (HTTP) o attraverso browser, come Internet Explorer o Mozilla Firefox
- Richieste GET e POST: GET è il metodo con cui vengono richieste la maggior parte delle informazioni ad un Web server, tali richieste vengono veicolate tramite query string, cioè la parte di un URL che contiene dei parametri da passare in input ad un'applicazione. Il metodo POST, invece, consente di inviare dati ad un server senza mostrarli in query string, è ad esempio il caso dei form.

Architettura del Sistema

L'architettura del sistema sarà composta da:

- Lato utente con un interfaccia con cui l'utente può interagire con il bot, questa sarà una semplice chat di Telegram dove l'utente tramite pulsanti e messaggi di testo potrà interagire con il bot
- Lato server dove il sistema elaborerà l'input dato, farà una richiesta alla pagina web ricerca lezioni fornita dall'Università di Perugia e infine elaborerà l'output ricevuto per poterlo comunicare all'utente tramite la chat Telegram

Per visualizzare i vari diagrammi cliccare nel link presente nel titolo di ogni diagramma

Diagramma di Classe

Per la realizzazione del diagramma di classe ho pensato a quali "entità" facessero parte del progetto e le loro relazioni. Ho pensato a 8 classi, non tutte sono esplicitamente definite nel codice del progetto, in quanto non ho usato una programmazione ad oggetti, ma in futuro per implementare ulteriori funzionalità ho in mente di riprogettare il codice ad oggetti, esplicitando la maggior parte di queste classi,per ora alcune sono "implicite" nel funzionamento del progetto ma ho preferito metterle comunque per una maggiore comprensione dell'architettura del progetto e delle relazioni tra le componenti.
Le classi sono:

- **Dipartimento**: Ogni oggetto della classe è corrispettivamente un dipartimento dell'università di perugia, con i suoi vari attributi e con una lista di corsi appartenti a quel dipartimento.
  Essendo il dipartimento composto da corsi e il corso non può esistere senza il dipartimento, ho scelto una relazione di tipo "composizione"
- **Corso**: Ogni oggetto della classe è rispettivamente un corso offerto dall'Università di Perugia, oltre alle sue caratteristiche, ha un attributo idDipartimento per stabilire sotto quale Dipartimento è svolto il corso e infine contiene una lista di lezioni.
  Come nel caso precedente si può fare un ragionamento analogo per la relazione tra corso e lezione, perciò ho scelto una relazione di tipo "composizione"
- **Lezione**: Ogni oggetto della classe è rispettivamente una lezione svolta nel corso indicato dall'attributo idCorso, che a sua volta farà parte di un dipartimento, oltre alle caratteristiche del corso, conterrà il link all'aula Teams in cui si svolge la lezione e due attributi idDocente1 e idDocente2 per indicare i 2 docenti che svolgono la lezione, se questa è svolta da un solo docente, l'attributo idDocente2 sarà null, per questo ho inserito come valore di default null.
  Questa volta tra Lezione e Docente è presente una relazione di tipo "associazione" in quanto nessuno dei due è composta da più parti dell'altro e le due possono esistere indipendetemente dall'altra, la relazione si può descrivere come 1 o 2 docenti svolgono 0 o più lezioni (un docente potrebbe non svolgere lezioni temporaneamente)
- **Docente**: Ogni oggetto della classe è rispettivamente un docente dell'Università di Perugia, questo conterrà una lista delle lezioni da lui svolte
- **Risultato**: L'elemento centrale del diagramma di classi, ogni oggetto della classe indica un risultato della ricerca svolta dal bot e comunicata all'utente tramite la chat.

Questo ha una relazione di “associazione” con tutte e 4 le classi descritte in precedenza, in quanto 0 o più risultati contengono ciascuno 1 dipartimento, 1 corso, 1 lezione e 1 o 2 docenti.

- **Utente**: L’altro elemento centrale del diagramma, ha alcuni attributi come dipartimento, corso, tipoCorso e annoCorso per poter implementare la funzione di creare un proprio profilo personale in modo da permettere al bot di trovare tutte le lezioni che l’utente deve seguire in quell’anno e inviargli in unico messaggio tutti i link alle lezioni.
  Un altro attributo chiamato filtroCorso per permettere l’implementazione della funzione filtraCorso, con la quale i risultati vengono filtrati per un determinato corso.
  Infine contiene un attributo idChat che indica l’ id della chat con il Bot Telegram, questo lo mette in associazione con la classe Chat, ho scelto una relazione di tipo “associazione” con molteplicità 1 a 1, in quanto ad ogni utente è associata un unica chat e viceversa
- **Chat**: Ogni oggetto della classe rappresenta la chat su cui l’utente interagisce con il bot e in cui il bot scrive i risultati da lui cercati, questa non svolge l’algoritmo di ricerca ma si può vedere come una periferica di input/output con cui utente e bot possono interagire.
  La Chat è in relazione di “composizione” con la classe Risultato, la chat è composta da più risultati e il risultato non può esistere senza una chat.
- **Bot**: Questa classe istanzia un unico oggetto, il bot vero e proprio, ovviamente nel codice non è presente come classe in quanto è il codice stesso il bot, ma per una maggiore comprensione dell’architettura del progetto ho preferito metterlo.
  Questo contiene un token identificativo e un attributo elencoChat che rappresenta la lista di tutte le chat aperte con i vari utenti che usufruiscono del servizio, è quindi in relazione di “associazione” con la classe chat, un unico bot gestisce 0 o più chat, si potrebbe usare anche una composizione ma non sarebbe particolarmente adatta, in quanto il bot è un’entità indipendente, non è composta da più chat e la chat potrebbe esistere anche se il bot smettesse di funzionare.
  Infine è presente una relazione di “dipendenza” tra Bot e Risultato, in quanto il bot “usa” uno o più risultati come valore di ritorno dalla funzione ricercaLezione

Diagramma Caso d’Uso

Caso d’uso: UC1 – Ricerca Lezione
Attore primario (e altri attori): Utente, **Bot**, Chat ed eventuali Risultati
Precondizioni: L’utente non è in possesso del link per la lezione desiderata
Postcondizioni: L’utente è in possesso del link per la lezione desiderata
Scenario principale:

1. L’utente accede al sistema
2. L’utente seleziona il comando “ricerca lezioni”
3. L’utente inserisce la parola o parole chiavi con cui effettuare la ricerca
4. Il bot la ricerca attraverso la pagina web di unipg
5. Il bot restituisce all’utente tutti i risultati dalla ricerca effettuata

Scenario secondario:
a)Nel caso in cui l’utente inserisca una parola chiave scorretta o ricerca una lezione non esistente

1. Il bot ricerca nella pagina web con la parola data

2. Il bot si accorge di non aver trovato nessun risulato
3. Il bot fornisce all'utente un messaggio con cui gli comunica la mancanza di risultati
4. Il bot permette all'utente di fare una nuova ricerca con la parola giusta

b)Nel caso in cui l'utente quando gli viene richiesta la parola, invece che testo invia un qualsiasi altro tipo di messaggio

1. Il bot si accorge subito che non è un messaggio di tipo testo, prima ancora di fare la ricerca
2. Il bot invia all'utente un messaggio di errore, informandolo che sono ammessi solo messaggi di tipo testo
3. Il bot permette all'utente di fare una nuova ricerca con il messaggio del giusto tipo

Nel diagramma ho inserito un associazione tra l'utente e i 3 casi d'uso presenti nel progetto, l'utente seleziona il caso d'uso, allo stesso modo è presente un associazione tra bot e i casi d'uso, il bot esegue i casi d'uso.
Inoltre c'è una relazione di inclusione tra due casi d'uso, lezioni personali include ricerca lezioni, in quanto ogni sua istanza esegue almeno una ricerca lezioni.
Questo perchè una volta trovati tutte le lezioni di un utente, il bot ne ricerca il link uno per uno tramite ricerca lezioni.

Diagramma di Sequenza e di Collaborazione

Nei due diagrammi ho rappresentato il comportamento relativo allo scenario del caso d'uso "RicercaLezioni".
Per farlo ho utilizzato vari gruppi di oggetti:

- **Utente**: Per far iniziare la sequenza invia un comando alla chat, il comando RicercaLezioni, la Chat una volta ricevuto invia un messaggio di ritorno chiedendo la parola con cui cercare, a quel punto l'utente invia un messaggio alla chat contenente la parola da cercare
- **Chat**: La chat un volta ricevuto il comando e la parola da cercare lo invia al bot per processarlo con un messaggio sincrono e si mette in attesa di un messaggio di ritorno contenente i Risultati della ricerca
- **Bot**: Una volta ricevuto il comando con la parola, fa uno WebScraping della parola sulla PaginaWeb contenente i link delle lezioni
- **PaginaWeb**: Invia un messaggio di ritorno contenente i risultati della ricerca
- **Risultato**: Una volta che il bot riceve i risultati crea un oggetto Risultato per ogni risultato inviato e li invia con un messaggio di ritorno alla chat, la quale a sua volta (implicitamente) li ritorna all'utente.

Diagramma di Stato

Descrive il comportamento del sistema per tutti i casi d'uso cioè per ogni comando inviato dall'utente.
Lo stato iniziale indica lo stato in cui l'utente è nella chat.
L'utente effettua una transizione inviando un comando alla chat, ci si sposta allo stato "Comando Inviato", successivamente il bot elabora il comando e ci si sposta nello stato "Comando Elaborato".

Da questo stato ci sono due possibili strade, una se il bot trova un errore nell'invio del comando, come ad esempio la richiesta di un comando non esistente o un parametro sbagliato, si va nello stato "Comando Sbagliato", da qua il bot ritorna un messaggio di errore e si arriva allo stato "Esito Comando Restituito".
L'altra strada invece viene attraversata se il bot non trova errori, quindi ci si sposta in "Comando Giusto", esegue il comando, come ad esempio la ricerca della lezione, si sposta in "Comando Eseguito" e infine arriva allo stato in comune con l'altra strada "Esito Comando Restituito" ritornando il risultato del comando.
Da quest'ultimo stato si passa allo stato finale dove l'utente ha visualizzato il risultato del comando richiesto.

Diagramma di attività

Descrive la logica procedurale del caso d'uso RicercaLezioni.
Nello stato iniziale l'utente è nella chat con il bot.
L'utente **invia il comando** ricercalezioni, il bot riconosce il comando e gli **richiede la parola** da inserire e l'utente **invia la parola**.
A questo punto il bot controlla che la parola inserita sia corretta, se non sono presenti **errori** esegue la **ricerca con la parola** nella pagina web, se la ricerca ha **risultati**, il bot **restituisce i risultati**, altrimenti restituisce un messaggio segnalando l'**assenza di risultati**.
Se invece sono presenti errori, il bot **rifiuta il comando** e **restituisce l'errore** per cui ha rifiutato il comando.
A questo punto il comando è terminato, l'utente ha ottenuto l'esito del comando ed il bot è pronto a svolgere un nuovo comando e si passa quindi allo **stato finale**.

Implementazione

Per l'implementazione del progetto ho utilizzato Python 3.9 e utilizzando la libreria telepot che supporta le API di Telegram e la libreria beautifulSoup.
Ho utilizzato un interfaccia a pulsanti per scegliere il comando e poi tramite messaggi richiedo i vari parametri per la ricerca o per il profilo.
Per implementare la funzione ricercaLezioni, il bot prende il comando, prende il parametro con cui cercare, tramite Web Scraping fa una richiesta POST alla pagina web, questa restituirà la pagina contenente i risultati.
A questo punto ricerco nella pagina risultante tutti gli elementi con tag <tr> e li salvo uno per uno in una variabile, la quale verra poi scomposta nelle varie colonne che verranno salvate in degli array contenenti le varie parti del risultato (nome, docente, corso e link).
Una volta trovate tutte le righe e divise per colonne si stampano i risultati al'utente.

Per la ricerca filtrata si utilizza lo stesso procedimento, ma una volta trovati i risultati, si stampano solo quelli in cui la colonna Nome Corso contiente il filtro impostato.

Infine per la ricerca delle lezioni personali il procedimento è un pò più complesso e lungo.
Il bot riceve il comando e i vari parametri per capire a quale corso appartiene o studente.
Per trovare esattamente il corso svolto e quindi le lezioni presenti in questo, faccio una richiesta GET alla pagina impostando come parametri della ricerca il nome del corso e il tipo di corso inserito dall'utente.
Da questa richiesta verrà fornita una pagina contente i vari corsi trovati, da questi risultati si

prende solo il corso che come colonna Dipartimento ha il dipartimento dall'utente selezionato.
Una volta trovato il corso entra nella scheda formativa del corso, va nella sezione Insegnamenti e prende la colonna Nome di tutti i risultati che fanno parte della prima,seconda o terza tabella in base all'anno di corso scelto, si salva tutti i nomi delle lezioni trovate in un array e infine richiama la funzione ricercaLezioni per ogni elemento dell'array.

Link al Bot: https://t.me/RicercaLezioniUniPg_Bot
Link a codice Bot in HTML:
https://drive.google.com/file/d/1HHQXVZpT7_6ew7FXwYIlsfosg_nvzPh7/view?usp=sharing

Test Funzionali

Prenderemo in considerazione la funzione Lezioni Personali in quanto è quella che richiede più input e quindi vincoli.
I vincoli sono:

- L'input "dipartimento" e "corso" di appartenenza, saranno una stringa di almeno 1 lettera e massimo 30 lettere
- L'input "tipo di corso" sarà un intero nell'intervallo [0,3], 0: Triennale, 1: Magistrale, 2: Magistrale Ciclo Unico 5 anni, 3: Magistrale Ciclo Unico 6 anni
- L'input "anno corso" sarà un intero nell'intervallo [1,3] nel caso di corso triennale, [1,2] in caso di magistrale…

Parametro Input	Classe Valida	Classe Non Valida	Classe Non Valida
Dipartimento	0<len(Dipartimento)<=30	len(Dipartimento)<1	len(Dipartimento)>30
Dipartimento Valori Estremi	len(Dipartimento)=1,30	len(Dipartimento)=0	len(Dipartimento)=31
Corso	0<len(Corso)<=30	len(Corso)<1	len(Corso)>30
Corso Valori Estremi	len(Corso)=1,30	len(Corso)=0	len(Corso)=31
Tipo Corso	0<=Tipo Corso<=3	-MAXINT<=Tipo Corso<0	3<Tipo Corso<=MAXINT
Tipo Corso Valori Estremi	0,3	-1	4
Anno Corso (Triennale)	1<=Anno Corso<=3	MAXINT<=Anno Corso<1	3<Anno Corso<=MAXINT
Anno Corso (Triennale) Estremi	1,3	0	4
Anno Corso (Magistrale)	1,2	MAXINT<=Anno Corso<1	2<Anno Corso<=MAXINT
Anno Corso (Magistrale) Estremi	1,2	0	3

Dipartimeno	Corso	Tipo Corso	Anno Corso	Validità e copertura
Matematica	**Informatica**	**0**	**2**	Si, copre tutte le classi valide
‘’ (stringa vuota)	Informatica	0	2	No, estremo len(dip.)>0
len(dip)=31	Informatica	0	2	No, estremo len(dip.)<=30
Matematica	**‘’ (stringa vuota)**	0	2	No, estremo len(corso)>0
Matematica	**len(corso)=31**	0	2	No, estremo len(corso)<=30
Matematica	Informatica	**-1**	2	No, estremo Tipo Corso>0
Matematica	Informatica	**4**	2	No, estremo Tipo Corso<=3
Matematica	Informatica	0	**-1**	No, estremo Anno Corso>0
Matematica	Informatica	0	**4**	No, estremo Anno Corso<=3
Matematica	Informatica	1	**-1**	No, estremo Anno Corso>0
Matematica	Informatica	1	**3**	No, estremo Anno Corso<=2

Con il primo test si coprono tutte le classi valide, in quanto si deve cercare di coprire con il minimo numero di test, più classi possibili.
Mentre per le classi non valide va fatto un test per ogni classe non valida, quindi si usano i valori subito prima o subito dopo quelli estremi per testare la non validità delle classi.
Sono 5 classi valide coperte con 1 solo test e sono 10 classi non valide coperte con 10 test, per un totale di 11 casi di test Blackbox con analisi dei valori estremi.
Inoltre ho svolto ulteriori test manualmente, richiamando più volte le varie funzioni in ordini diversi e con input diversi per verificare la correttezza del codice.

Design Pattern

Un Design Pattern che secondo me è applicabile al progetto è il modello architetturale ModelViewController.
Il model sarebbe il risultato della ricerca, indipendente da come viene rappresentato nella chat e indipendente da che modalità o funzione viene ricavato.
View sarebbe la chat, cioè la rappresentazione del model sullo schermo, possono esistere viste multiple dello stesso modello, ad esempio con più risultati in un unico messaggio, con un messaggio per ogni risultato o ad esempio restituendo i risultati in un file PDF.
Il Controller sarebbe il bot, che riceve gli input dell’utente, li traduce in richieste di servizio e le esegue. L’utente interagisce esclusivamente con il bot.
Questa struttura permetterebbe di disaccoppiare le viste dai modelli, in modo da permettere di modificare la rappresentazione dell’oggetto indipendentemente dalla logica e struttura del modello.

In questo modo si potrebbe utilizzare la stessa vista per rappresentare due modelli diversi, ad esempio con l'aggiunta della ricerca esami, si potrebbe rappresentare le lezioni e gli esami con la stessa vista.
Un altro Design Pattern potrebbe essere quello del Singleton, applicabile alla classe Bot in quanto si deve garantire l'esistenza di un unica istanza della classe Bot.