DESIRA chatbot - Manuale sviluppatore

2021

Contents

1 Introduzione

L'obiettivo di questo progetto è fornire una semplice interfaccia conversazionale per la creazione di segnalazioni relative a richieste di intervento nel contesto della mitigazione del dissesto idrogeologico. Il sistema è costituito da quattro componenti principali:

1. Un'istanza di Botpress per la realizzazione della logica di conversazione;
2. Un programma in Python che agevola la comunicazione tra l'istanza di Botpress e Telegram agendo da intermediario;
3. Un'istanza di MongoDB per la persistenza dei dati;
4. Un'interfaccia web per il monitoraggio delle segnalazioni da parte di operatori autorizzati.

Questo manuale fornisce dettagli sull'implementazione dei primi due e sull' interazione con il database.

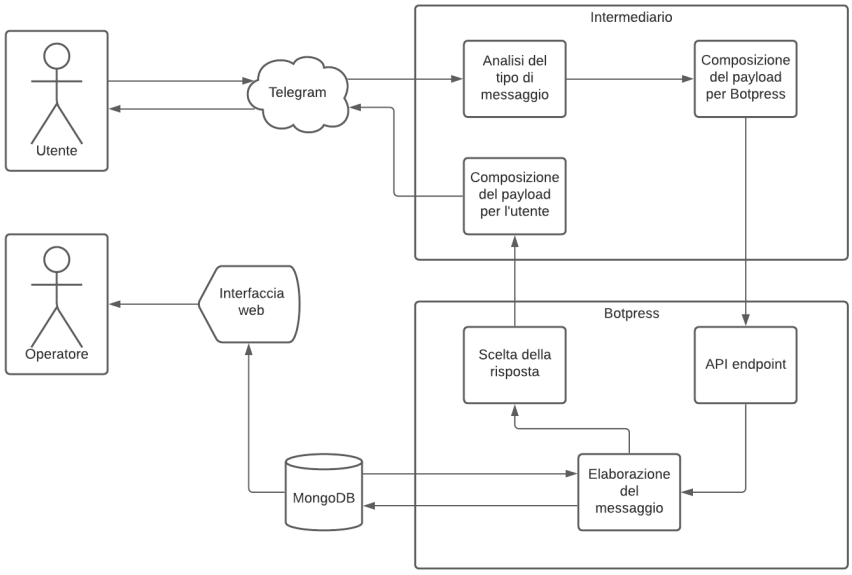

Fig. 1: Struttura generale del sistema

2 Implementazione

In questa sezione vedremo i dettagli dell'implementazione e le motivazioni dietro ad alcune scelte.

2.1 Logica di conversazione

La logica del chatbot è divisa in sei flussi principali:

- `Main`: flusso di entrata creato automaticamente, ogni conversazione inizia da qui;
- `Registration`: flusso per la registrazione di un nuovo utente;
- `New_report`: flusso per la creazione di una nuova segnalazione;
- `Edit_report`: flusso per la modifica di una delle proprie segnalazioni attive;
- `View_reports`: flusso per la visualizzazione delle proprie segnalazioni;
- `Delete_account`: flusso per la cancellazione dell'account.

Poichè i grafi diventano rapidamente illeggibili dopo un certo numero di nodi, i flussi più complessi, ovvero `Registration, New_report` e `Edit_report`, sono stati spezzati in sottoflussi, visionabili nelle rispettive cartelle `subflows`. In questi casi, il flusso principale passa temporaneamente il controllo della conversazione al sottoflusso desiderato, che al suo termine effettua un salto ad uno specifico nodo del suddetto flusso principale, restituendogli il controllo.

Oltre a quelli fin'ora menzionati, figurano altri tre flussi:

- `Error handling`: creato automaticamente, nel nostro caso è il flusso a cui si salta in caso di errori. Il salto a questo flusso avviene tramite una flow-wide transition con condizione `temp.error == true`, presente in tutti i flussi in cui viene utilizzata una delle funzioni che impostano la suddetta variabile quando si verifica un errore.
- `Timeout`: se l'utente invia un messaggio dopo lo scadere del timeout definito in `data/global/botpress.config.json`, la conversazione salta immediatamente a questo flusso. Questo salto è gestito automaticamente da Botpress.
- `Too_many_errors`: poichè i nodi scelta presentano un numero massimo di retry per quando l'utente non sceglie una delle opzioni proposte, ad esempio scrivendo in chat anzichè selezionare una delle opzioni a schermo, è stato necessario introdurre un flusso a cui ripiegare nel caso in cui il suddetto massimo venga superato. In questo caso, il flusso non fa altro che notificare l'utente del fatto che sono stati commessi troppi errori e riportare la conversazione all'inizio, tornando al flusso `Main`.

Struttura della cartella dei flussi:

```
Flows
├── Main
├── Error handling
├── Timeout
├── view_reports
│   └── View_reports
├── registration
│   ├── subflows
│   │   ├── Ask_secondary_contact
│   │   ├── Ask_primary_contact
│   │   └── Ask_personal_info
│   └── Registration
├── new_report
│   ├── subflows
│   │   ├── Add_photos
│   │   ├── Add_location
│   │   └── Add_description
│   └── New_report
├── edit_report
│   ├── subflows
│   │   ├── Edit_photos
│   │   ├── Edit_location
│   │   └── Edit_description
│   └── Edit_report
├── delete_account
│   └── Delete_account
└── Too_many_errors
```

2.1.1 Main

Quando un utente avvia una conversazione, il bot controlla che questo sia registrato con una chiamata a `check_user.js`, che effettua una query al database e imposta la variabile `temp.registered` di conseguenza. A seconda del risultato viene deciso se offrire all'utente la possibilità di registrarsi o se mostrare la lista delle operazioni possibili.

Questo flusso presenta anche un nodo utilizzato come punto di entrata secondaria: `should_continue`. Alcuni flussi, raggiunto il loro termine, effettuano un salto a questo nodo, in cui si chiede l'utente se ha ancora bisogno di qualcosa o meno, riproponendo la lista delle operazioni in caso di risposta positiva. Questo approccio è stato scelto dopo aver escluso le due alternative: chiudere la sessione dopo ciascuna operazione, opzione scartata per evitare di obbligare l'utente a inviare ogni volta un nuovo messaggio per riaprire un'altra sessione; riproporre automaticamente la lista al termine di ogni operazione, con il rischio di incalzare inutilmente l'utente quando ha terminato di utilizzare il bot.

2.1.2 Registration

La registrazione è divisa in tre sottoflussi:

- `Ask_primary_contact`: in questo flusso si chiede all'utente di specificare in che modo preferirebbe essere contattato in caso di necessità, scegliendo se fornire un indirizzo e-mail o un numero di telefono. Il contatto lasciato dall'utente è controllato nella forma da un'espressione regolare nel caso dell'indirizzo e-mail o tramite la libreria `libphonenumber_js` nel caso del numero di telefono.

- `Ask_secondary_contact`: se l'utente decide di inserire un contatto secondario, da utilizzare nel caso quello primario risulti irraggiungibile, il controllo passa a questo flusso. Le modalità sono perlopiù identiche a `Ask_primary_contact`.

- `Ask_personal_info`: in questo flusso si chiede all'utente di fornire cognome e nome. Viene effettuato un controllo con `check_if_text.js`, che verifica se il messaggio è effettivamente testo, verificando l'assenza dei tag aggiunti dall'intermediario nel caso di dati speciali. Per maggiori informazioni su questi tag visionare la sezione 2.2.

Il flusso termina con una richiesta di conferma dei dati e la conseguente aggiunta dell'utente al database. L'identificativo univoco associato a ciascun utente è l'ID a lui assegnato da Telegram.

2.1.3 Add_report

La creazione di una nuova segnalazione è divisa in tre sottoflussi:

- `Add_location`: questo flusso implementa il primo passo per la creazione di una segnalazione, ovvero la richiesta di comunicazione del luogo in cui è avvenuto l'evento. All'utente viene proposta una scelta tra l'invio di coordinate attraverso l'apposita funzionalità di Telegram o l'invio di una descrizione testuale del luogo. In entrambi i casi l'input viene controllato, verificando che vi sia il tag `[$COORDS]` nel primo caso e che non sia presente nessun tag nel secondo.

- `Add_description`: semplice flusso per la richiesta di una descrizione dell' evento. Anche in questo caso si verifica che la risposta sia composta esclusivamente da testo.

- `Add_photos`: se l'utente decide di aggiungere foto alla segnalazione, il controllo passa a questo flusso. Quando una foto viene ricevuta, la funzione `download_photo.js` la scarica dall'URL generato da Telegram e la inserisce nell'array `temp.photos`, dopodichè viene chiesto se si vogliono inviare altre foto, eventualmente ripetendo l'operazione.

Come nel caso della registrazione, il flusso termina mostrando all'utente un riepilogo dei dati inseriti, chiedendo conferma e inviando la segnalazione al

database. L'identificativo univoco associato a ciascuna segnalazione di un particolare utente è la data di creazione della stessa, nella forma `DD/MM/YY, hh:mm:ss`.

2.1.4 Edit_report

La modifica di una segnalazione inizia inviando all'utente un riepilogo delle sue segnalazioni attive, eventualmente notificandolo nel caso in cui non ne abbia. Segue l'invio di una lista di opzioni in cui si chiede all'utente di scegliere la segnalazione da modificare, utilizzando la data come identificativo. La scelta è generata programmaticamente in `choose_report.js` perchè i nodi scelta possono essere usati solo quando conosciamo a priori tutte le opzioni, cosa che in questo caso non avviene, poichè dipendono dal risultato della query. Scelta una segnalazione, si effettua una nuova query al DB per recuperarne le foto, che vengono salvate nell'array `temp.photos`, per poi iniziare il loop di modifica chiedendo all'utente cosa intende modificare. A questo punto entrano in gioco tre sottoflussi a cui si salta quando viene scelta la rispettiva opzione:

- `Edit_location`: flusso per la modifica del luogo associato alla segnalazione. Le modalità di acquisizione sono affini a quelle utilizzate in fase di creazione di una nuova segnalazione.
- `Edit_description`: flusso per la modifica della descrizione dell'evento. Come per la posizione, la struttura del flusso richiama quella utilizzata durante la creazione.
- `Edit_photos`: il flusso inizia con un controllo su `temp.photos` per verificare se vi sono foto associate alla segnalazione. In caso di risultato positivo, le foto vengono salvate localmente nella cartella indicata nell'hook `after_bot_mount/initial_config.js` con nome nella forma `ID_INDICE` utilizzando l'ID dell'utente e l'indice della foto nell'array, incrementato di uno (per evitare che partano da zero). Il salvataggio in locale è necessario per permettere all'intermediario di inviare le foto all'utente, per maggiori informazioni visionare la sezione 2.2. Dopo aver salvato localmente le foto, queste vengono mostrate all'utente, ciascuna con il proprio indice (sempre incrementato di uno) in didascalia, e viene poi chiesto se si intende eliminarne qualcuna o aggiungerne di nuove. L'aggiunta di nuove foto avviene con le stesse modalità di `Add_photos`. L'eliminazione di una foto consiste nell'inviare all'utente una scelta le cui opzioni sono gli indici incrementati di uno, aggiungendo all'array `temp.to_delete` l'indice scelto dall'utente, proponendo infine la possibilità di confermare, continuare o annullare l'operazione. In caso di conferma, tutte le foto il cui indice si trova nel suddetto array vengono rimosse da `temp.photos`.

Il flusso di modifica termina quando l'utente decide di confermare le modifiche, di annullarle o di eliminare la segnalazione.

2.1.5 View_reports

Il flusso di visualizzazione delle segnalazioni consiste nel chiedere all'utente di scegliere se visionare tutte segnalazioni, solo quelle attive oppure solo quelle archiviate, per poi effettuare una query appropriata in base alla scelta. Nel caso in cui non vi siano segnalazioni del tipo specificato viene mostrato un messaggio speciale, in caso contrario ne viene presentato un riepilogo. La query al database è effettuata chiedendo tutte le segnalazioni che hanno un valore del campo `status` che corrisponda al tipo richiesto. Al momento viene semplicemente controllato se il campo ha valore "Archiviata" o meno.

2.1.6 Delete_account

La cancellazione consiste nell'eliminare dal database, previa conferma, il documento associato all'utente. Le segnalazioni non vengono cancellate e, nel caso in cui l'utente decida di iscriversi nuovamente, vengono nuovamente assegnate a lui (a patto che l'ID Telegram sia lo stesso).

2.2 Invio di messaggi non testuali

L'intermediario in Python è stato introdotto per permettere l'invio di messaggi speciali, come immagini o coordinate generate da Telegram, a Botpress, che accetta solo payload testuali.

2.2.1 Foto

Quando l'utente invia una foto al bot, l'intermediario compone un messaggio di testo per Botpress nella forma

```
[$PHOTO]URL
```

dove `URL` è l'URL temporaneo fornito da Telegram per il download dell'immagine dai propri server. Il bot deve processare questo tipo di messaggio utilizzando un'azione apposita, `Util/download_photo.js`, che riconosce il prefisso `[$PHOTO]`, scarica la foto e la inserisce in un array salvato nella variabile `temp.photos`.
L'invio di foto dal bot all'utente è gestito in maniera diversa: il bot salva le immagini da inviare nella cartella specificata durante la configurazione nell'hook `after_bot_mount/initial_config.js` e invia all'intermediario un messaggio nella forma

```
[$PATHS]PATH1|PATH2|PATH3|...|PATHN
```

contenente i percorsi delle immagini salvate, separati dal carattere '`|`'. Come già menzionato in precedenza, il nome del file è dato da una concatenazione dell'ID dell'utente, un underscore e la posizione della foto nell'array. L'intermediario riconosce il prefisso `[$PATHS]`, legge le foto da file, le invia a Telegram e le cancella dal disco.

2.2.2 Coordinate

Nel caso in cui l'utente voglia inviare delle coordinate al bot tramite l'apposita funzionalità di Telegram, l'intermediario compone un messaggio nella forma

```
[$COORDS]LATITUDINE|LONGITUDINE
```

che deve essere interpretato dal bot con la funzione `Util/get_coordinates.js`, il cui funzionamento è simile alle funzioni descritte sopra.

2.2.3 Nodi scelta

L'aggiunta dell'intermediario ha anche introdotto la necessità di implementare manualmente la conversione del payload generato da Botpress per un nodo scelta in una forma che fosse comprensibile a Telegram. Questo è gestito dall'intermediario che, riconosciuto il payload come scelta, itera sull'array delle opzioni e costruisce una `ReplyMarkupKeyboard` con un bottone per ciascuna di esse.

2.2.4 Altri tipi di dati speciali

Immagini, coordinate e nodi scelta sono gli unici tipi di payload speciali che l'intermediario è attualmente in grado di gestire. Per lo scambio di eventuali altri tipi di dati speciali, come video o documenti, sarà necessario estenderlo con funzionalità simili a quelle appena descritte.

2.3 Comunicazione tra i componenti

Vediamo brevemente in che modo i vari componenti del sistema comunicano tra loro.

2.3.1 Intermediario - Telegram

L'intermediario legge il token del bot Telegram da un file `token.txt`, che deve trovarsi nella stessa directory del file `botpress_middleman.py`, e lo utilizza per comunicare con Telegram. La comunicazione avviene attraverso una libreria, `python-telegram-bot`, che mette a disposizione metodi di alto livello per lo scambio di messaggi con bot Telegram. Più nel dettaglio, l'intermediario registra un `MessageHandler` che si occupa di chiamare la funzione `handle_message` ogni volta che viene rilevato, tramite polling, l'arrivo di un messaggio al bot.

2.3.2 Botpress - Intermediario

Per comunicare con Botpress, l'intermediario utilizza un'API messa a dispozione da Botpress stesso che permette di inviare messaggi attraverso richieste POST e di leggerne le repliche nel payload della risposta. La funzione `forward` costruisce un payload testuale per Botpress in base ai contenuti del messaggio, come

descritto nella sezione 2.2, e lo invia con una richiesta POST effettuata utilizzando il modulo `requests`. L'URL base ha la seguente forma, supponendo che Botpress sia esposto su `http://localhost:3000`:

```
http://localhost:3000/api/v1/bots/report-handling/converse/
```

Alla fine di questo URL viene aggiunto l'ID Telegram dell'utente per conto del quale stiamo inoltrando il messaggio, ottenibile con opportune funzioni della libreria `python_telegram_bot`.

2.3.3 Botpress - Database

La comunicazione con il database avviene esclusivamente attraverso Botpress utilizzando l'apposito driver di MongoDB per Node.js.

3 Schema dei dati

Il database contiene due collezioni di documenti: `users` e `reports`. L'utilizzo di MongoDB, un database NoSQL basato su documenti, ci ha permesso di non definire uno schema rigido, facilitando eventuali modifiche alla rappresentazione dei dati.

3.1 Users

Questa collezione contiene un documento per ogni utente nella seguente forma:

```
{
    id,
    first_name
    last_name,
    primary_contact,
    secondary_contact
}
```

Il campo `secondary_contact` è facoltativo.

3.2 Reports

Questa collezione contiene un documento per ogni segnalazione nella seguente forma:

```
{
    user_id,
    date,
    photos,
    description,
    status,
    location,
```

```
    location_desc
}
```

I campi `location` e `location_desc` sono mutuamente esclusivi e il campo `photos` è facoltativo.