Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

ISW Bot: sviluppo di un bot Telegram come integratore di strumenti di collaborazione agile

Relatore:
Chiar.mo Prof.
Paolo Ciancarini

Presentata da:
Marco Dong

Terza Sessione
2020-2021

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Avere mezzi di comunicazione che siano efficienti ed efficaci è essenziale per ogni organizzazione, specialmente in questo periodo di pandemia di COVID-19 dove le quarantene e il distanziamento sociale sono all'ordine del giorno.

Nel contesto dello sviluppo del software, la comunicazione e la collaborazione sono dei fattori critici, in quanto permettono di rispettare le scadenze affinchè si consegni un software di qualità, e di limitare lo spreco delle risorse in modo da tenere alto il livello organizzativo del team.

La programmazione è spesso considerata puramente un'attività di risoluzione di problemi. Questo è vero per il programmatore solitario, ma in un ambiente di team, l'attività principale deve essere la collaborazione. Il Covid-19 ha visto portare una transizione senza precedenti al lavoro a distanza, pertanto una comunicazione efficace risulta indispensabile.

Secondo quanto riportato dal sondaggio [11], questa mutazione ha avuto il risultato che l'80% degli individui hanno riscontrato una vera difficoltà nel lavorare proprio per via dell'impedimento nella comunicazione.

In un ufficio condiviso, tutti sono a conoscenza delle conversazioni a voce alta, oltre ad essere liberi di esaminare la lavagna. A casa, ognuno è al proprio computer nella proprie stanza. Al di fuori dell'utilizzo di uno strumento come Slack o Zoom, gli sviluppatori non sono totalmente consapevoli di ciò che sta facendo il resto del loro team. Una soluzione è utilizzare strumenti di collaborazione e canali di

comunicazione, nonostante esistano un'infinità, occorre però che le informazioni non siano frammentate e che siano facilmente reperibili.

Pertanto questo progetto ha lo scopo di affrontare i problemi di comunicazione studiando tre software open source di collaborazione, integrandoli nella piattaforma Telegram attraverso lo sviluppo di un Bot.

Capitolo 2

I tool di collaborazione open source

Gli strumenti di collaborazione sono app, programmi software o piattaforme che aiutano team o individui a semplificare il processo creativo e a collaborare in modo più efficace ed efficiente. Consentono a manager e dipendenti di assegnare compiti, aggiornarsi sui progressi, riferire sui risultati e in generale migliorare i flussi di lavoro e la comunicazione, sia internamente che esternamente.

Le categorie più diffuse di strumenti di collaborazione sono le seguenti:

- comunicazione — *Teams, Slack, Chanty, Mattermost...*
- videoconferenza — *Teams, Zoom, Google Meet...*
- condivisione di file su cloud — *Dropbox, OneDrive...*
- collaborazione su documenti — *Google Suite, Office 365...*
- gestione progetti — *Trello, Jira, Asana, Taiga...*
- sviluppo — *Github, BitBucket, Gitlab...*

Si andrà quindi ad esaminare i seguenti tre strumenti:

- Taiga — https://www.taiga.io/

- Gitlab — https://about.gitlab.com/
- Mattermost — https://mattermost.com/

2.1 Taiga

Taiga è uno strumento open source per gestire progetti, dedicato a un team Agile. Ha un'interfaccia intuitiva e semplice, ma allo stesso tempo è completo di tutte le funzionalità di cui un team ha bisogno.

Per iniziare occorre un account sulla piattaforma. Creando un progetto è possibile scegliere un template tra Scrum e Kanban, o duplicare un progetto già esistente da Taiga o importare da altre piattaforme come Asana e Trello.

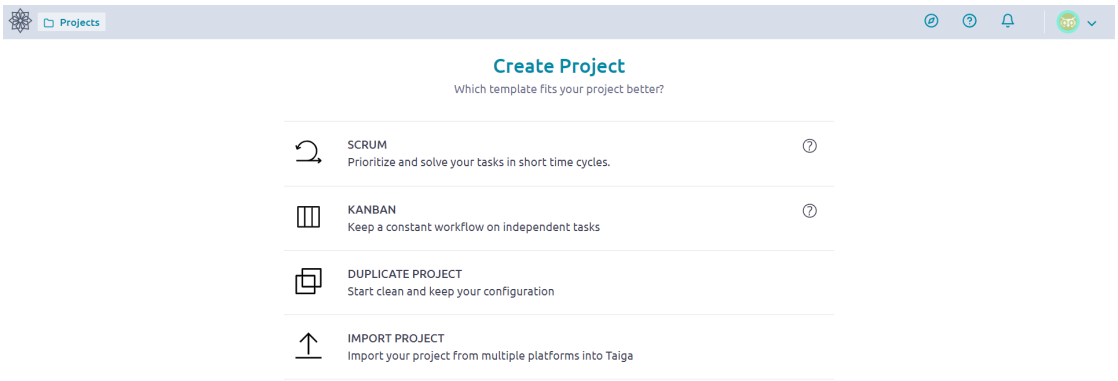

Figura 2.1: Creazione progetto in Taiga

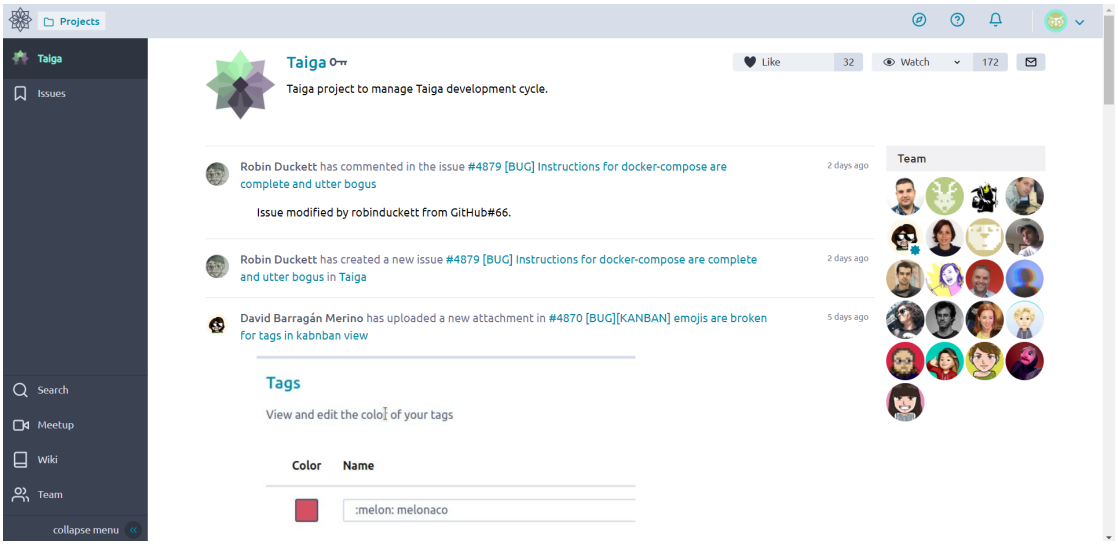

Figura 2.2: Dashboard di Taiga

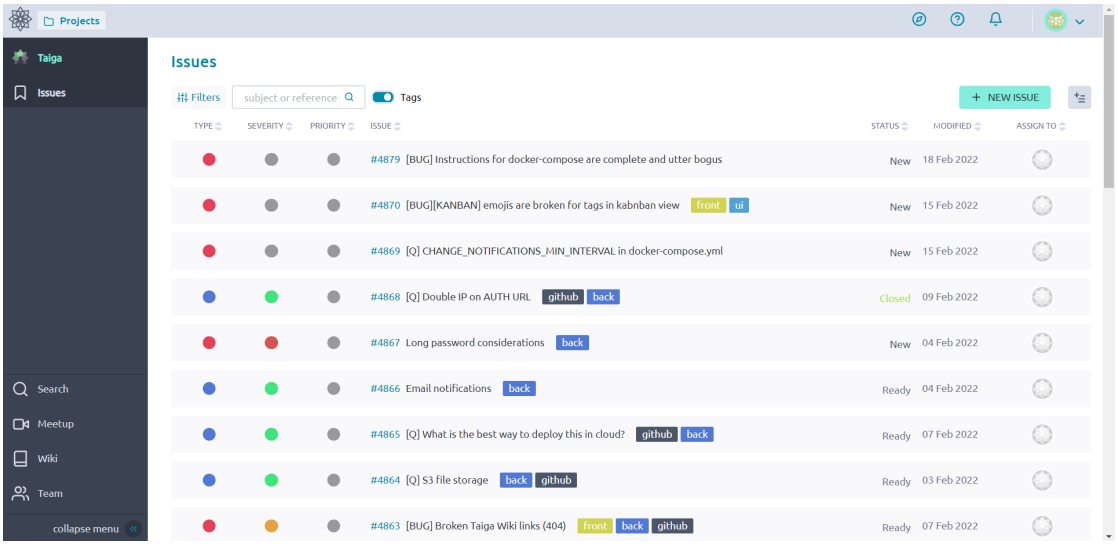

Figura 2.3: Schermata di Issues di Taiga

2.2 Gitlab

Gitlab è uno strumento di sviluppo open source che vanta della sua ricchezza di funzionalità completamente dedicata al DevOps che serve a lavorare in modo più

semplice ed efficiente, a consegnare software più velocemente e ridurre al contempo il rischio e costi su un progetto. Diverse organizzazioni importanti come Nvidia, Goldman Sachs e Siemens si affidano a questa piattaforma proprio per questi motivi.

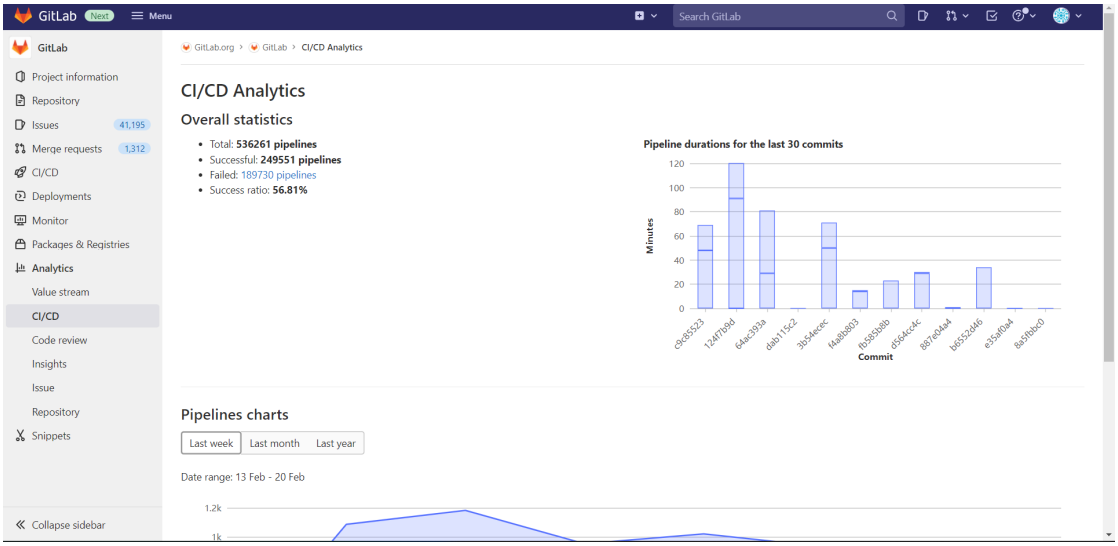

Figura 2.4: Pagina di Analytics del progetto in Gitlab

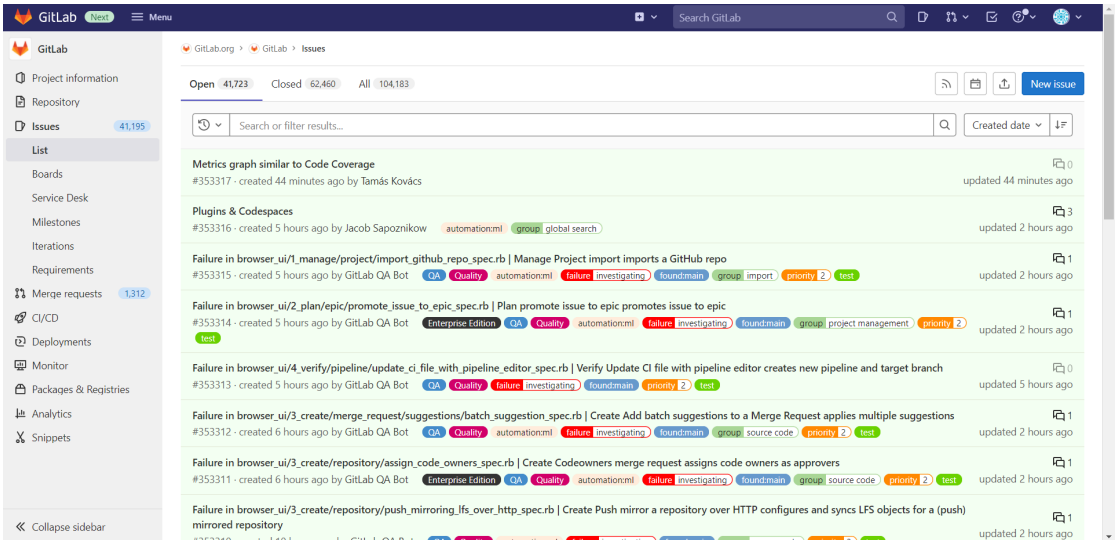

Figura 2.5: Issues del progetto Gitlab

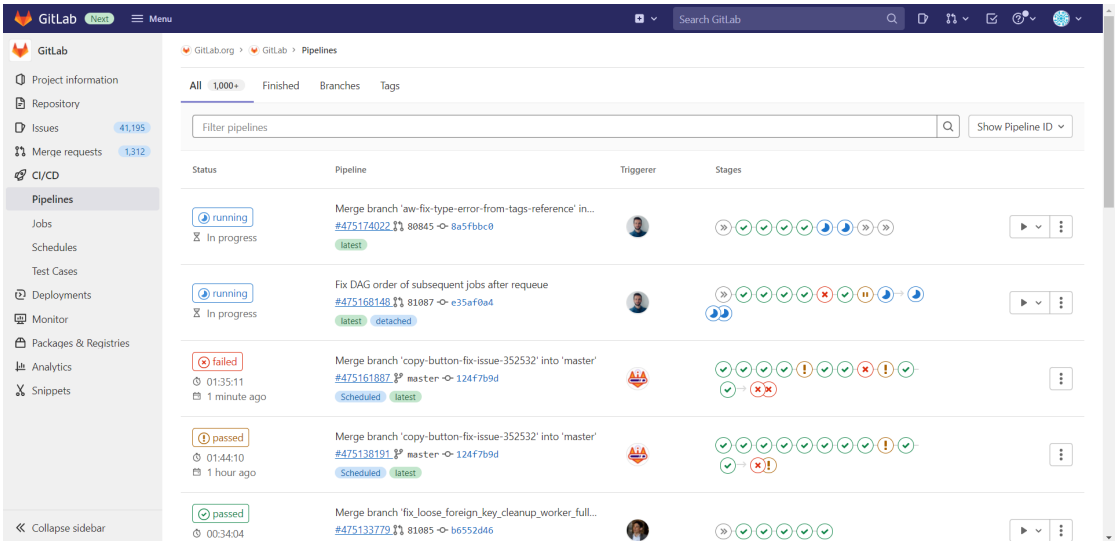

Figura 2.6: Pipelines del progetto Gitlab

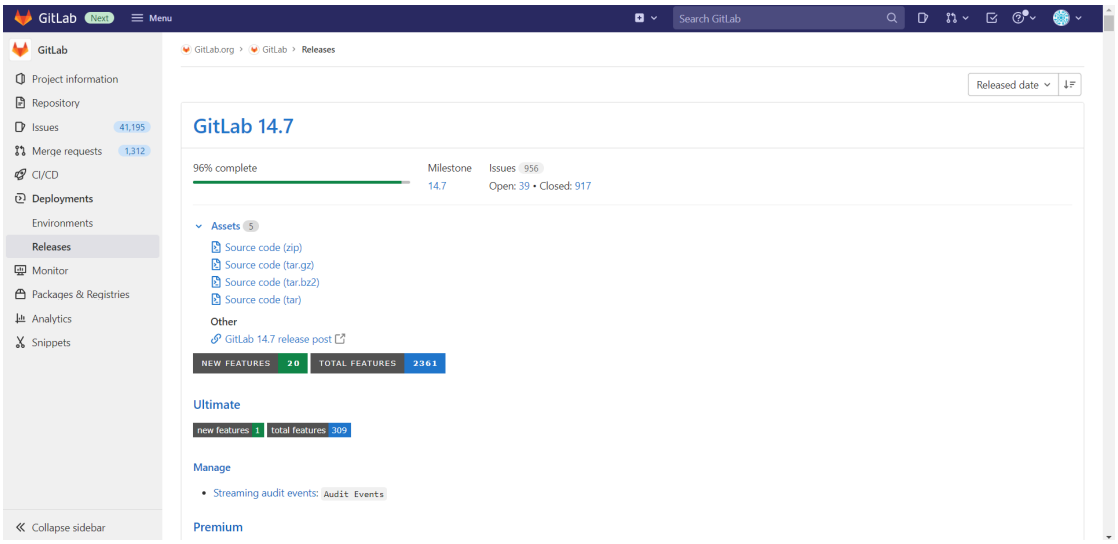

Figura 2.7: Releases del progetto Gitlab

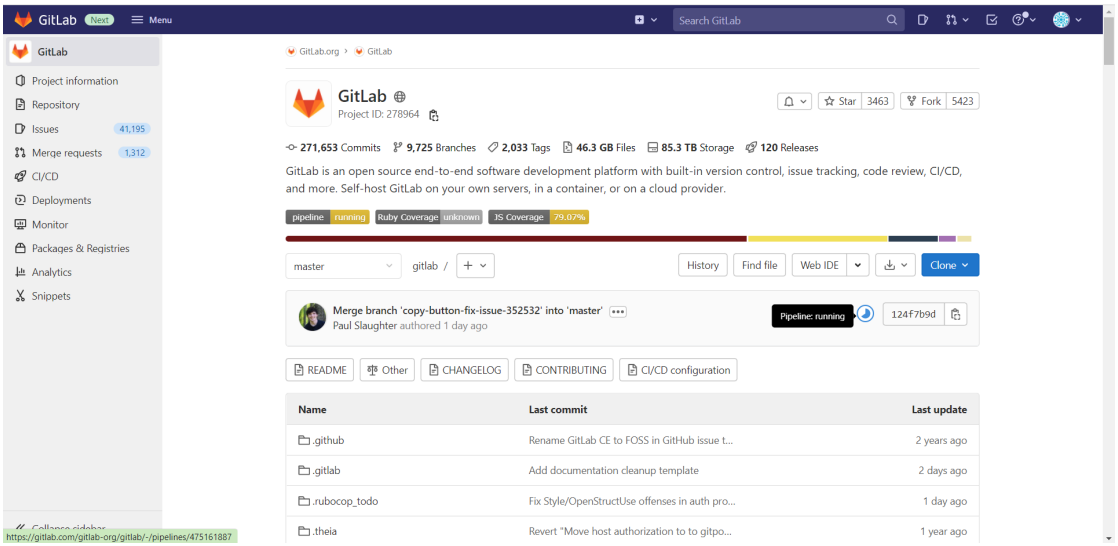

Figura 2.8: Repo del progetto Gitlab

2.3 Mattermost

Mattermost è uno strumento che fa parte della categoria dei tool di comunicazione, anch'esso open source, il suo grande vantaggio è di poterlo usare su un server privato e di poter integrare varie piattaforme di terze parti usando i Webhook in entrata.

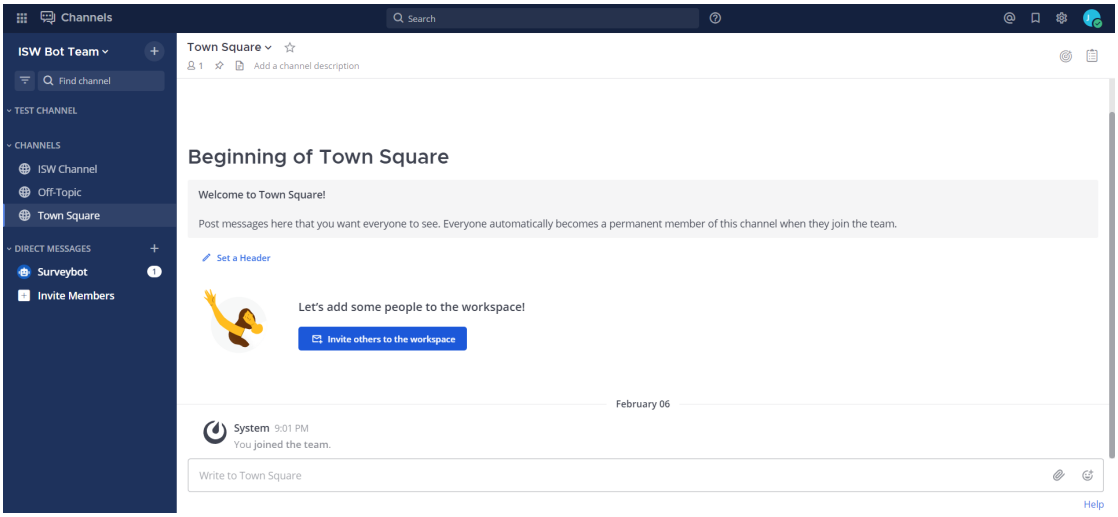

Figura 2.9: Schermata principale di Mattermost

Capitolo 3

Telegram

Telegram è una popolare app di messaggistica multipiattaforma basata su cloud con versioni dell'app disponibili per iOS, Android, Windows, Mac e Linux. Ha anche una versione Web. Questa è ampiamente utilizzata perché offre alcune funzionalità avanzate di privacy e crittografia end-to-end, supporta funzionalità di chat di gruppo di grandi dimensioni e canali di comunicazione. Inoltre, non ha legami con altre piattaforme di social media, il che rende il servizio più attraente per alcune tipologie di persone.

A differenza di altre applicazioni di comunicazione, Telegram consente di condividere un numero illimitato di foto, video e file (doc, zip, mp3, ecc. . .) fino a 2 GB ciascuno.

La privacy e la sicurezza di Telegram sono considerate affidabili perché l'API del servizio e tutte le applicazioni client sono open source e disponibili per la valutazione e l'integrazione da parte di qualsiasi sviluppatore.

E' disponibile anche un'API Bot, una piattaforma per sviluppatori che consente a chiunque di creare facilmente strumenti specializzati per Telegram, integrare qualsiasi servizio e persino accettare pagamenti da utenti di tutto il mondo.

3.1 I bot Telegram

I Bot sono dei piccoli programmi che girano all'interno di Telegram. Sono creati da sviluppatore di terze parti utilizzando le API Telegram Bot.

I Bot sono come profili utenti normali infatti è possibile direttamente dialogare con loro o aggiungerli in gruppi. Ogni Bot deve possedere almeno le seguenti informazioni:

- Nome del Bot
- Username utilizzato per menzioni e per ottenere il link
  - 5-32 caratteri latini case-insensitive, numeri o underscore
  - deve avere alla fine la stringa 'bot'
- token auto-generato utilizzabile per effettuare richieste API

3.1.1 Privacy

Il Bot all'interno di un gruppo può assumere due diverse modalità:

- Privacy Mode OFF
- Privacy Mode ON

Quando la modalità privacy è disabilitata, il Bot potrà solamente vedere messaggi che hanno un'utilità per lui:

- Comandi esplicitamente destinati a lui (es. **/command@bot**).
- Comandi generici da utenti (es. **/start**) se il Bot è stato l'ultimo ad inviare il messaggio
- Tutti i messaggi inviati dal Bot
- Tutti i messaggi degli utenti che rispondono al messaggio del Bot

Nella modalità Privacy Mode ON, attivabile solamente dall'amministratore del gruppo, il Bot potrà vedere tutti i messaggi disponibili ad esclusione di quelli degli altri Bot, in modo da prevenire loop non graditi.

3.1.2 BotFather

BotFather è il Bot con il quale si creano e si modificano i Bot, si definisce il 'padre di tutti i Bot'. I comandi accettati dal padre sono diversi, ma quelli che possono tornare utili in questo progetto sono i seguenti:

- /newbot — crea un nuovo Bot
- /mybots — ritorna una lista di Bot con importazioni modificabili
- /setname — cambia il nome di un Bot
- /setdescription — cambia la descrizione di un Bot (possono essere utilizzati al più 512 caratteri). E' visibile dagli utenti all'inizio della conversazione con il Bot.
- /setabouttext — cambia l'informazione del profilo del Bot (al più 120 caratteri). Quando viene condiviso il link del Bot, questa informazione verrà visualizzata assieme.
- /setuserpic — cambia la foto profilo del Bot.
- /setcommands — aggiunge e modifica la lista di comandi disponibili del Bot (che sono modificabili dinamicamente anche dalle API).
- /deletebot — elimina il Bot
- /setjoingroups — abilita il permesso del Bot di essere aggiunto ai vari gruppi.
- /setprivacy — sceglie la modalità di privacy di default quando il Bot viene aggiungo a un gruppo.
- /token — genera un nuovo token utilizzato per autorizzare azioni associati a un Bot.
- /revoke — revoca il token di un Bot.

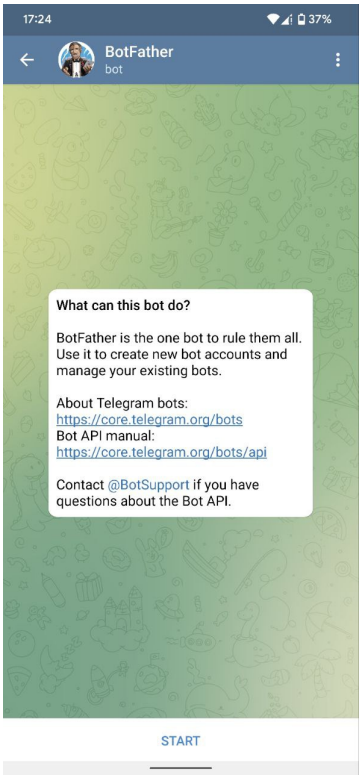

Figura 3.1: La chat di BotFather

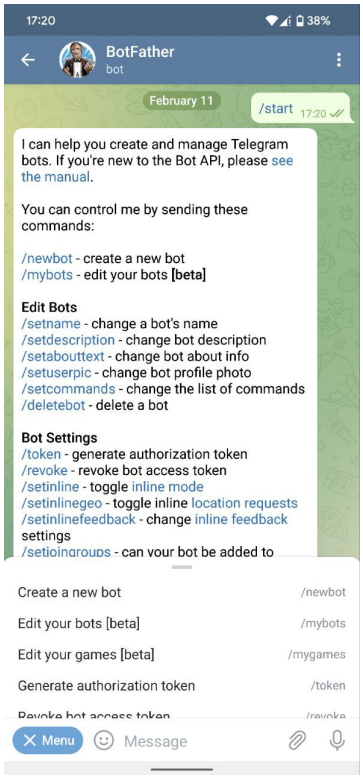

Figura 3.2: I comandi di BotFather

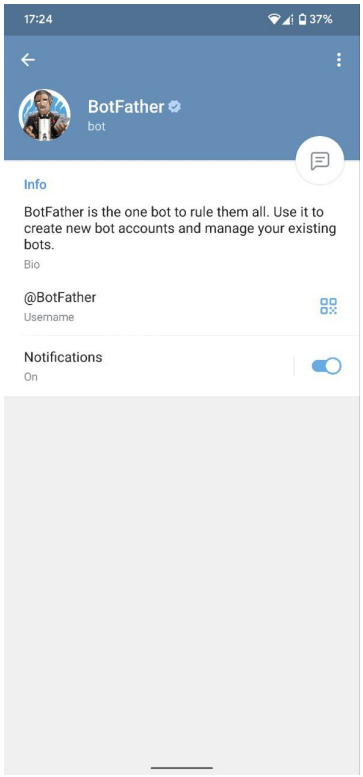

Figura 3.3: Il profilo di BotFather

3.2 Telegram bot API

3.2.1 Richieste

Requisiti

Le API di Telegram Bot vengono interfacciate tramite richieste HTTP. Per fare una richiesta occorrono alcuni requisiti da soddisfare:

- il token del Bot
- nome del metodo
- richiesta con codifica UTF-8

GET/POST

Gli eventuali parametri per richieste GET e POST devono essere passati in una di queste quattro modalità:

- Stringa di ricerca nell'URL (URL query string)
- *application/x-www-form-urlencoded*
- *application/json* (escluso il caso per fare l'upload di un file)
- *multipart/form-data* (per fare l'upload di file)

Return

La risposta ad una richiesta contiene un oggetto JSON, che può cambiare forma in base al fatto che sia positiva o negativa. In entrambi i casi avrà sempre un campo "ok" che rappresenta il fatto che la richiesta sia andata a buon fine e un campo opzionale "description" che descrive in linguaggio naturale il risultato ricevuto.

Ad una risposta positiva, il campo "**ok**" sarà uguale a **true** accompagnato dal campo "**result**" contenete l'oggetto della ricerca. Invece, ad una risposta negativa, il campo "**ok**" sarà uguale a **false** e saranno presenti i campi "**description**" e "**error_code**" un tipo di valore intero che aiuta a gestire l'errore.

Errors

Gli errori, quando si lavorano con le API, devono essere gestiti correttamente, sono composti dal codice, dal tipo dell'errore e da una serie di parametri che variano in base all'errore. I codici che possono assumere sono le seguenti:

- 303 SEE_OTHER
- 400 BAD_REQUEST
- 401 UNAUTHORIZED
- 403 FORBIDDEN
- 404 NOT_FOUND
- 406 NOT_ACCEPTABLE
- 500 INTERNAL

Figura 3.4: Richiesta a Telegram Bot API

3.2.2 Metodologie di comunicazione

Esistono due metodi mutuamente esclusivi per ottenere aggiornamenti dal Bot: uno attraverso la tecnica del Long Polling e l'altro utilizzando il metodo del Webhook, in qualsiasi caso l'oggetto ritornato sarà una lista di tipo **Update** serializzato in JSON.

Update

La tabella di seguito rappresenta l'oggetto **Update**. A parte *update_id*, ci può essere al più un altro campo allo stesso tempo.

Campo	Tipo	Descrizione
update_id	Integer	identificatore univoco di valore positivo sequenziale
message	Message	un nuovo messaggio in arrivo (testo, immagine, sticker...)
edited_message	Message	nuova versione del messaggio che era noto al bot
channel_post	Message	un nuovo messaggio in arrivo all'interno di un canale (testo, immagine, sticker...)
edited_channel_post	Message	nuova versione del messaggio che era noto al bot all'interno di un canale
inline_query	InlineQuery	arrivo di una richiesta inline
chosen_inline_result	ChosenInlineResult	la scelta che è stato fatto in una richiesta inline
callback_query	CallbackQuery	l'arrivo di una richiesta callback
shipping_query	ShippingQuery	l'arrivo di una richiesta di spedizione
pre_checkout_query	PreCheckoutQuery	l'arrivo di una richiesta di ordine
poll	Poll	stato del sondaggio
poll_answer	PollAnswer	cambiamento del voto in un sondaggio non anonimo
my_chat_member	ChatMemberUpdated	cambiamento dello stato del bot all'interno di un gruppo
chat_member	ChatMemberUpdated	lo stato un membro di un gruppo è stato cambiato (il bot deve essere amministratore del gruppo)
chat_join_request	ChatJoinRequest	alla ricezione di una richiesta di partecipazione del gruppo (il bot deve avere il permesso *can_invite_users*)

Tabella 3.1: I campi del tipo **Update**

Long Polling

La tecnica del *Long Polling* è una variazione del *polling*. Per definizione, il *polling* avviene quando un programma chiede continuamente dei dati o delle risorse che possono essere disponibili o meno. Il *Long Polling* fa la stessa cosa ma ci si aspetta che i dati o le risorse richiesti non ci siano al momento in cui lo si chiedono, per cui ci si mette in attesa per un breve periodo. In questo contesto, il client chiede a Telegram se ci sono aggiornamenti, usando la tecnica del *Long Polling*, se non ci sono aggiornamenti, Telegram lascia aperta la connessione per un certo periodo, ritornando una lista di aggiornamenti, se ci sono, altrimenti una lista vuota allo scadere del tempo. Questa metodologia di comunicazione con Telegram è vantaggiosa quando non si ha a disposizione un server esposto, potendo di conseguenza utilizzare anche con un computer personale e testare velocemente il Bot.

getUpdates

Parametro	Tipo	Descrizione
offset	Integer	un intero che rappresenta *update_id* di un **Update** con valore maggiore di tutti quelli precedentemente ricevuti. Verrà quindi applicato un offset nell'array di ritorno (se è un valore negativo, l'offset partire dalla fine dell'array)
limit	Integer	deve essere un valore compreso tra 1-100 (default è 100), limita il numero di elementi
timeout	Integer	valore che rappresenta la finestra di secondi per cui la connessione resta aperta
allowed_updates	Array di String	un array di tipi di **Update** che si vuole filtrare

Tabella 3.2: Parametri opzionali da passare all'API **getUpdates**

Webhook

A differenza del *Polling*, sarà a cura del server comunicare se ci sono aggiornamenti attraverso una richiesta HTTP POST verso un indirizzo IP o dominio pubblico. Il vantaggio di quest'ultimo è proprio di evitare il continuo "hai aggiornamenti?" sprecando risorse, soprattutto caso in cui gli aggiornamenti non sono frequenti.

setWebhook

Parametro	Tipo	Descrizione
url	String	l'url su cui inviare la richiesta (se è vuota rimuove l'integrazione con il Webhook)
certificate	InputFile	la chiave pubblica del proprio certificato
ip_address	String	utilizza un indirizzo IP statico invece di risolverlo tramite un DNS
max_connections	Integer	numero di connessioni https al server (1-100, default 40)
allowed_updates	Array di String	un array di tipi di **Update** che si vuole filtrare

Tabella 3.3: Parametri opzionali da passare all'API **setWebhook**

deleteWebhook

Parametro	Tipo	Descrizione
drop_pending_updates	Boolean	tutte gli **Update** in attesa vengono eliminati

Tabella 3.4: Parametri opzionali da passare all'API **deleteWebhook**

getWebhookInfo E' senza parametri, torna un tipo **WebhookInfo** se esiste l'integrazione Webhook, altrimenti un oggetto con un campo *url* vuoto.

WebhookInfo

Parametro	Tipo	Descrizione
url	String	url del Webhook (vuoto se non è stato impostato)
has_custom_certificate	Boolean	se è stato usato un certificato
pending_update_count	Integer	numero di **Update** ancora in attesa di essere inviate
ip_address	String	l'indirizzo IP usato per il Webhook
last_error_date	Integer	la data in unix time dell'ultimo errore
last_error_message	String	messaggio in linguaggio naturale dell'ultimo errore avvenuto
max_connections	Integer	numero di connessioni https al server (1-100, default 40)
allowed_updates	Array di String	un array di tipi di **Update** che si vuole filtrare

Tabella 3.5: I campi di **WebhookInfo**

Capitolo 4

Lo sviluppo del Bot in Python

Python è un linguaggio di programmazione interpretato di alto livello. La sua filosofia è la leggibilità del codice utilizzando indentazioni al posto delle parentesi graffe per definire blocchi. I suoi costrutti linguistici e il suo approccio orientato agli oggetti mirano ad aiutare i programmatori a scrivere codice chiaro e logico per progetti su piccola e larga scala. Generalmente Python viene utilizzato per sviluppare applicazioni distribuite, scripting e computazione numerica.

4.1 Python Telegram Bot

Esiste una libreria in Python denominata **python-telegram-bot**, che rende disponibile un'encapsulazione dell'utilizzo delle API di Telegram con semplici classi, metodi e rappresentazioni in classi Python di tutti i tipi disponibili nella documentazione ufficiale, come per esempio **Update** e **WebhookInfo**.

La libreria ha il requisito di avere la versione di Python ≥ 3.6 suddividendosi in due package, di seguito verranno poi elencati i metodi e i campi principalmente più utili in questo contesto per alcuni tipi.

Esistono vari metodi per installare una libreria Python (chiamata anche modulo), il metodo tradizionale, nonchè il più semplice, è aver installato il package installer di Python chiamato **pip**. Per installare il modulo basta eseguire la seguente linea da terminale:

```
pip install python-telegram-bot
```

nota: In alcuni sistemi con specifici settaggi, occorre utilizzare al posto di **pip**, **pip3** per riferirsi all'uso di Python 3.

4.1.1 Il package "telegram"

In questo package sono contenuti tutti i tipi e i metodi che sono nella documentazione ufficiale di Telegram:

```
import telegram
```

telegram.Bot

Questa classe consente di far eseguire azioni al Bot, ovvero contiene tutti i metodi supportati dall'API di Telegram. Il costruttore della classe deve almeno ricevere il token del Bot per operare.

- *approve_join_request(user_id, timeout=None, api_kwargs=None)* → **bool** — accetta la richiesta di un utente di entrare in gruppo o canale

- *decline_chat_join_request(chat_id, user_id, timeout=None, api_kwargs=None)* → **bool** — rifiuta la richiesta di un utente di entrare nel gruppo o canale
- *send_message(chat_id, text, parse_mode=None, disable_web_page_preview=None* → **telegram.Message** — il metodo è usato per inviare un messaggio avendo la chat id
- *edit_message_text(text, chat_id, message_id)* → **telegram.Message** — modifica un messagio all'interno di una chat
- *delete_message(chat_id, message_id, timeout=None, api_kwargs=None)* → **bool** — elimina un messaggio che era stato inviato meno di 48h fa. Se è in un gruppo con privilegi amministratori, può eleminare qualsiasi messaggio.
- *get_updates(offset=None, limit=100, timeout=0, read_latency=2.0, . . .)* → **List[telegram.Update]** — Questo metodo ottiene gli aggiornamenti pendenti da Telegram usando la tecnica del Long Polling
- *set_webhook(url=None, certificate=None, . . .)* → **bool** — Il metodo definisce che la comunicazione deve usare il webhook
- *delete_webhook(timeout=None, api_kwargs=None, drop_pending_updates=None)* → **bool** — elimina un webhook che era stato integrato
- *get_webhook_info(timeout=None, api_kwargs=None)* → **telegram.WebhookInfo** — Ottiene le informazioni del webhook
- *get_me(timeout=None, api_kwargs=None)* → **telegram.User** — Ottiene semplici informazioni del bot, utile per testare per testare il bot
- *get_my_commands(timeout=None, api_kwargs=None, scope=None, language_code=None)* → **List[telegram.BotCommand]** — questo metodo ottiene la lista dei comandi del bot

telegram.Update

Oltre a parametri e campi che rappresentano lo stesso della tabella 3.1 e costanti per fare il pattern matching per identificare il tipo di update, ci sono campi e metodi aggiuntivi:

- *de_json(data, bot)* → **Telegram** — converte un JSON a un oggetto **Telegram**
- *effective_chat* → **Telegram.Chat** — utile per ottenere l'id di una chat a cui è riferito l'update e rispondere
- *effective_message* → **Telegram.Message** — che contiene il messaggio associato all'update
- *effective_user* → **Telegram.User** — , l'utente che ha fatto scattare l'update

telegram.Chat

Alcuni parametri variano in base al tipo di chat e possono pure non esserci, sono presenti costanti per fare il pattern matching per identificare il tipo di chat:

- *id* → **int** — identificatore unico corrispondente alla chat (con un utente, gruppo o canale)
- *type* → **str** — tipo di chat (privato con utente, gruppo o canale)
- *title* → **str** — titolo del gruppo o del canale
- *username* → **str** — l'username dell'utente che sta comunicando con il bot

telegram.Message

- *message_id* → **int** — identificatore univoco di un messaggio
- *from_user* → **telegram.User** — l'utente che ha scritto il messaggio
- *date* → **datetime.datetime** — data del messaggio

- *chat* → **telegram.Chat** — la chat in cui si trova il messaggio
- *text* → **str** — testo del messaggio lungo al più 4096 caratteri

telegram.User

- *id* → **int** — l'identificatore univoco del bot o del user
- *is_bot* → **bool** — se l'utente in questione è un bot
- *username* → **str** — l'username dell'utente
- *bot* → **telegram.Bot** — l'istanza del bot

4.1.2 Il package "telegram.ext"

In questo package sono contenute le classi e i metodi personalizzati che aiutano gli utenti a interagire facilmente con le API di Telegram

```
import telegram.ext
```

telegram.ext.Updater

Questa classe sfrutta la classe **telegram.ext.Dispatcher** per fornire un 'frontend' alla classe **telegram.Bot**, in modo tale da semplificare la gestione del Bot al programmatore.

Al suo costruttore, per poterlo gestire, occorre almeno che gli venga fornito il token del Bot o la sua istanza di **telegram.Bot**

- *bot* → **telegram.Bot** — l'istanza del bot usato in questo **Updated**
- *dispatcher* → **telegram.ext.Dispatcher** — un'istanza che consente di gestire l'invio e ricezione di **Update**
- *user_sig_handler* → **function** — una funzione callback che viene chiamata ogni volta che riceve una POSIX signal (es. SIGINT o SIGTERM)
- *running* → **bool** — indica se l'istanza è in esecuzione

- *start_polling(poll_interval=0.0, timeout=10, clean=None, ...)* → **Queue** — Il metodo inizierà continuamente a fare long polling aspettando degli **Update**
- *start_webhook(listen='127.0.0.1', port=80, url_path='', ...)* → **Queue** — Usando questo metodo invece starà in ascolto per le richieste POST da Telegram

telegram.ext.Dispatcher

La classe consente di inviare qualsiasi tipo di aggiornamento ai gestori di riferimento. Ad esempio quando si definisce un gestore per rispondere a dei comandi.

Si deve fornire al costruttore almeno un **telegram.Bot** e una **Queue**.

- *start()* → **None** — Fa partire il dispatcher in background
- *stop()* → **None** — ferma l'esecuzione del dispatcher
- *bot* → **telegram.Bot** — il **Bot** passato al gestore
- *update_queue* → **Queue** — la coda che contiene gli aggiornamenti
- *workers* → **int** — numero massimo di thread concorrenti per il Dispatcher
- *add_handler(handler)* → **None** — questo metodo aggiunge un **telegram.ext.Handler** da gestire
- *remove_handler(handler)* → **None** — il metodo rimuove la gestione di un **telegram.ext.Handler**

telegram.ext.Handler

Questa classe è quella che gestisce tutti gli aggiornamenti, viene principalmente usata per creare sottotipi in modo tale da coprirli in modo più specifico.

Al costruttore viene fornita una funzione di callback che viene invocata all'arrivo di un aggiornamento.

- *callback* → **Callable** — la funzione che gestisce l'aggiornamento
- *run_async* → **bool** — definisce se deve essere eseguito in modo asincrono

telegram.ext.CommandHandler

La classe gestisce i comandi, ovvero i messaggi che iniziano con / (slash) seguito da, obbligatorio quando ci sono più bot, @ (at) con l'username del bot:

```
/command[@bot_username]
```

Il costruttore deve ricevere il nome del comando e la funzione callback che lo gestisce.

- *command* → **telegram.utils.types.SLT[str]** — il nome del comando, il simbolo / (slash) non è da includere
- *callback* → **Callable** — il campo è la funzione di callback
- *allow_edited* → **bool** — se occorre gestire caso in cui il comando viene modificato successivamente

telegram.ext.MessageHandler

La classe gestisce i messaggi. I messaggi possono essere di tipo testo, foto, video, sticker...

Il costruttore deve ricevere almeno una funzione di callback.

- *filters* → **telegram.ext.filters** — il gestore considera solamente messaggi che passano questo filtro
- *callback* → **Callable** — il campo è la funzione di callback
- *allow_edited* → **bool** — se occorre gestire caso in cui il messaggio viene modificato successivamente

4.2 ISW Bot

ISW Bot è un Bot di Telegram che consente a un individuo o a un team di avere gli aggiornamenti dalle piattaforme Taiga, Gitlab e Mattermost in modo immediato, semplice e leggibile.

Se il team è in un gruppo, si aggiunge ISW Bot come membro, non ha bisogno di permessi speciali. Se si è un individuo lo si interagisce in privato. L'interazione con ISW Bot è lo stesso in ambo i casi.

Il Bot è trovabile tramite la ricerca globale di Telegram con l'username `@isw_cs_unibo_bot` e ha come nome ISW Bot.

ISW Bot, oltre a fornire gli aggiornamenti dalle piattaforme menzionate precedentemente, fornisce anche supporto per reperire informazioni riguardo al corso di Ingegneria del Software tenuto dal Professore Paolo Ciancarini.

Il codice sorgente completo è diviso in due branch: main e callback. Il branch main si occupa della gestione dei comandi del Bot ed usa il database Postgres. Il branch callback gestisce le chiamate POST da piattaforme esterne allocato sui server di Heroku. Il tutto comunque è reperibile alla seguente repository: https://gitlab.com/User16421775608429967892/isw-telegram.

4.2.1 I comandi

I comandi di ISW Bot possono essere invocati attraverso una lista di comandi con un apposito bottone denominato 'Menu' che è situato accanto al box per scrivere il messaggio nei dispositivi mobili (nelle applicazioni desktop e nella versione Z del browser); oppure attraverso una finestra a comparsa quando si digita / (slash).

- /help — spiega come poter interagire con il Bot mostrando la lista di comandi accettati, utile caso in cui non viene supportato il 'Menu' o la finestra della lista di comandi.

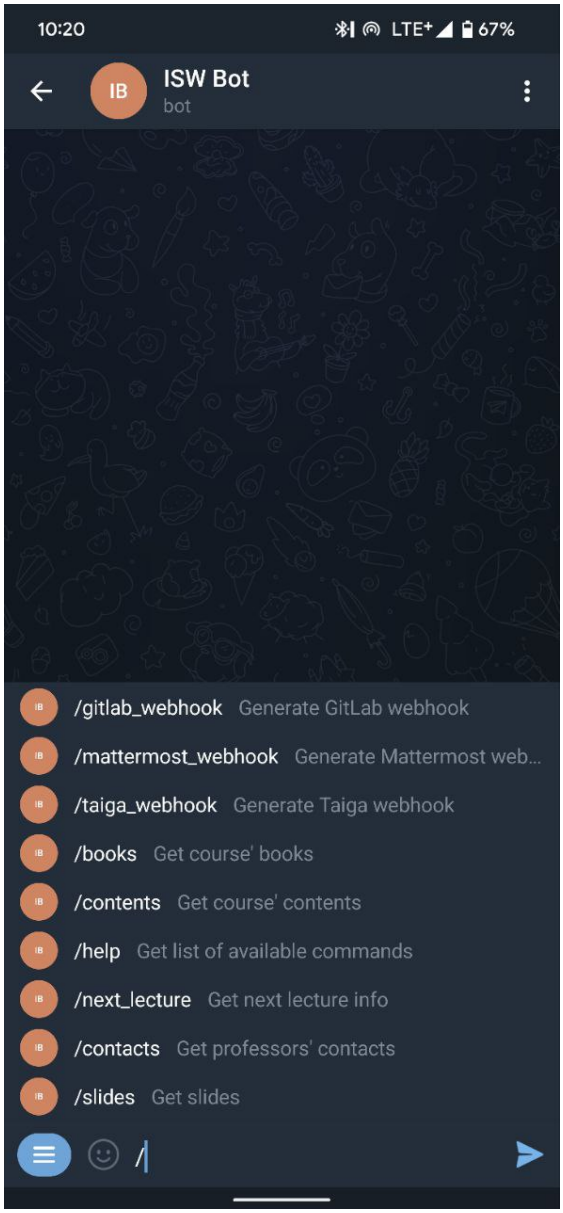

Figura 4.1: ISW Bot nell'applicazione Telegram per Android

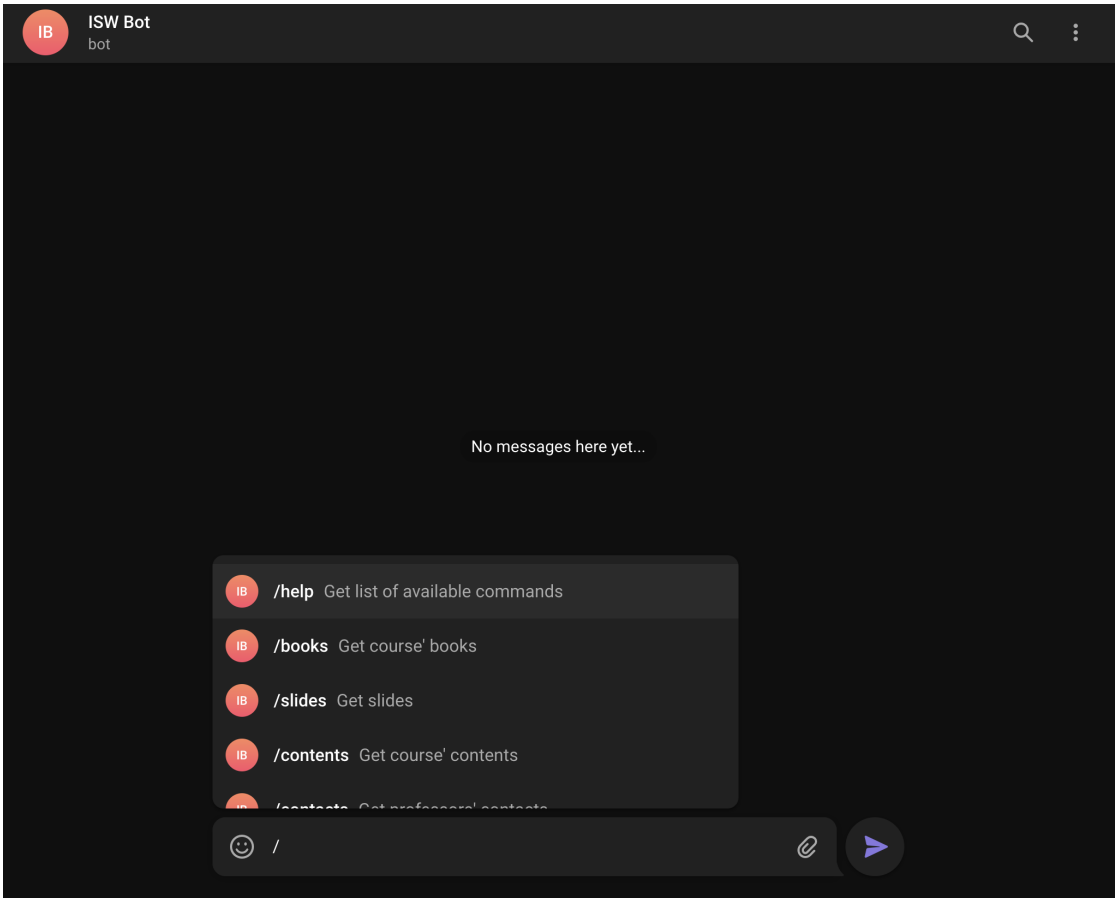

Figura 4.2: ISW Bot nella versione Z sul browser

Le informazioni sul corso

Le informazioni sul corso sono reperibili attraverso i seguenti comandi:

- /books — ritorna una lista di libri usati dal corso
- /contents — ritorna i contenuti che vengono spiegato nel corso
- /contacts — ritorna una lista di contatti dei professori
- /slides — ritorna le slide del corso
- /next_lecture — ritorna informazioni sulla lezione successiva

Alcuni di questi dati, come la lista di libri, i contenuti del corso e i contatti del professore, sono pressoché statici in quanto non vengono aggiornati spesso e si possono modificare manualmente quando occorre. In parte appartengono a questa categoria anche le slide, a differenza che queste sono legate con un link verso il sito del corso.

Le informazioni delle lezioni, invece, come data, orario e luogo vengono reperite utilizzando l'API dell'Università di Bologna.

Figura 4.3: ISW Bot il comando /contacts

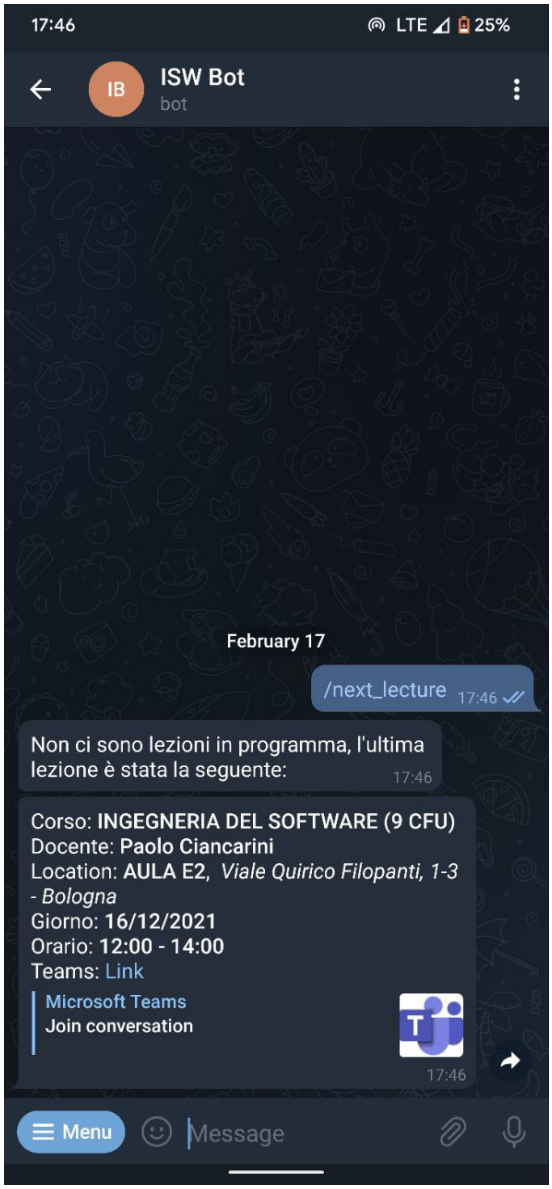

Figura 4.4: ISW Bot il comando /next_lecture

La seguente funzione è quella di callback che gestisce il comando */contacts*, la classe **Professor** è un'astrazione dell'oggetto in database chiamato **Professor**.

#isw-telegram/Telegram/Commands.py

```
from telegram import Update
from telegram.ext import CallbackContext

from Postgres.Object.Professor import Professor

def get_professor_contacts(update: Update, context: CallbackContext):
    for professor in Professor.get_all_professors(course_id):
        context.bot.send_message(chat_id=update.effective_chat.id,
                text=professor.formatted_contacts_text)
```

La seguente funzione, invece, è la funzione di callback che gestisce il comando */next_lecture* il quale utilizza la classe **Lecture** che astrae l'oggetto JSON tornato dall'API.

```
# isw-telegram/Telegram/Commands.py

from telegram import Update
from telegram.ext import CallbackContext

from Unibo.Lecture import Lecture

def get_next_lecture_info(update: Update, context: CallbackContext):
    lectures = Lecture.get_all_lectures()
    next_lectures = [lec for lec in lectures if lec.start_time >
                     datetime.now()]
    if len(next_lectures) > 0:
        next_lecture = next_lectures[0]
    else:
        context.bot.send_message(chat_id=update.effective_chat.id,
                                 text="Non ci sono lezioni in programma, "
                                      "l'ultima lezione è stata la seguente:")
        next_lecture = lectures[-1]
```

```
context.bot.send_message(chat_id=update.effective_chat.id,
                         text=next_lecture.get_lecture_md_formatted(),
                         parse_mode="Markdown")
```

Relazioni tra gli oggetti

Come database è stato utilizzato Postgres che è un database relazionale basato sul linguaggio SQL. Come ben noto, Postgres è affidabile, sicuro e scalabile, soprattutto è anche opensource.

Le relazioni all'interno del database viene mostrato di seguito, l'oggetto **Course** è il "genitore" di tutti gli altri.

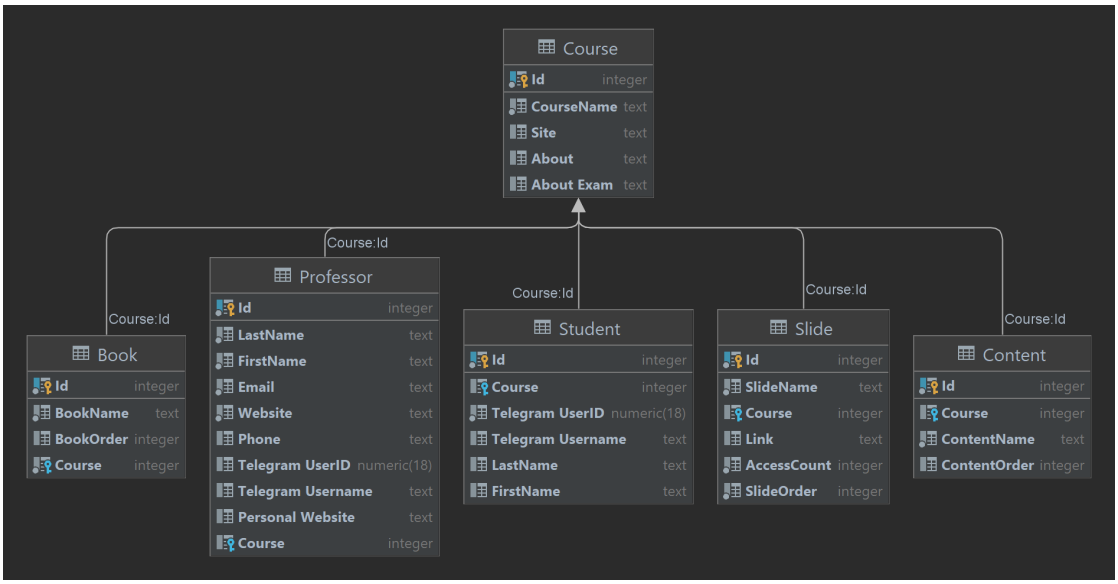

Figura 4.5: Diagramma delle relazioni degli oggetti Database

4.2.2 L'API di UNIBO

E' stato sfruttato l'utilizzo di una API esposta dall'Università di Bologna per implementare la funzione per ottenere i dati della lezione successiva. L'API non è ufficialmente documentata ma è stato possibile ottenerla dalle richieste HTTP fatte dal browser dall'indirizzo https://corsi.unibo.it/laurea/informatica/orario-lezioni.

Più nello specifico, il link relativo usato per ottenere l'oggetto JSON, partendo dall'indirizzo di base mostrato sopra, è il seguente: /@@orario_reale_json?AnnoCorso=3.

L'API consente di filtrare le lezioni attraverso il passaggio di parametri facoltativi come data e insegnamenti. Se non si specifica il parametro *AnnoCorso*, verrà considerato il valore 1.

In codice, è stato creato una classe che rappresenta l'oggetto JSON ritornato dall'API. E' possibile trovare il metodo statico della classe **Lecture** che consente di ottenere tutte le lezioni del corso ordinate in modo crescente per data e ora di inizio delle lezioni in modo tale da trovare velocemente la lezione successiva alla data attuale.

```
# isw-telegram/Unibo/Lecture.py

from datetime import datetime

import requests


class Lecture:
  def __init__(self, lecture_data):
    self.title = lecture_data.get("title")
    self.period = lecture_data.get("periodo")
    self.address = lecture_data.get("aule")[0].get("des_indirizzo")
    self.building = lecture_data.get("aule")[0].get("des_edificio")
    self.teams_link = lecture_data.get("teams")
    self.professor = lecture_data.get("docente")
    self.start_time = datetime.strptime(lecture_data.get("start"),
                "%Y-%m-%dT%H:%M:%S")
    self.end_time = datetime.strptime(lecture_data.get("end"),
                "%Y-%m-%dT%H:%M:%S")
    self.time = lecture_data.get("time")
```

```
@staticmethod
def get_all_lectures():
  """
  Il metodo ottiene tutte le lezioni del corso
  "ingegneria del software" disponibili
  dall'API di UNIBO ordinandole poi per data e ora di inizio
  in modo crescente
  :return: List[Lecture]
  """
  data = requests.get("https://corsi.unibo.it/laurea/informatica"
    "/orario-lezioni/@@orario_reale_json?anno=3")
  if data.status_code == 200:
    return sorted(
      [Lecture(lecture_data) for lecture_data in data.json()
        if "ingegneria del software" in
          lecture_data.get("title").lower()],
      key=lambda lecture: lecture.start_time)

def get_lecture_md_formatted(self) -> str:
  info = f"Corso: *{self.title}*\n" \
    f"Docente: *{self.professor}*\n" \
    f"Location: *{self.building}*,  _{self.address}_\n" \
    f"Giorno: *{self.start_time.strftime('%d/%m/%Y')}*\n" \
    f"Orario: *{self.time}*\n"
  if self.teams_link is not None:
    info += f"Teams: [Link]({self.teams_link})"
  return info
```

Ottenere le informazioni per configurare le integrazioni

I comandi sono i seguenti:

- /gitlab_webhook — spiega come poter integrare Gitlab
- /mattermost_webhook — spiega come poter integrare Mattermost
- /taiga_webhook — spiega come poter integrare Taiga

Ognuno di questi comandi genera un link dal formato `https://isw-bot.herokuapp.com/<tool>`

Si è ricorso all'ID della chat in modo tale da identificare facilmente chi è il gruppo o l'individuo che vuole avere gli aggiornamenti da queste piattaforme. La libreria Flask ha la funzionalità di gestire URL dinamici. La combinazione di ID e funzionalità di Flask permette di focalizzare il codice sulla parte dell'estrazione dei dati fino all'elaborazione di questi.

Le implementazioni delle loro integrazioni, sono basate secondo la loro documentazione ufficiale.

4.2.3 Integrazione Mattermost

L'integrazione di Mattermost consiste nel fare 'echo' dai canali applicando opportuni filtri direttamente dalla piattaforma.

Per iniziare, occorre generare il link da ISW Bot attraverso il comando /mattermost_webhook e nella sezione 'Outgoing Webhooks', accedibile da 'Integrations' in alto a sinistra tramite i nove quadratini, e inserire l'URL. Infine bisogna scegliere il 'Content Type' 'application/json'.

```
#isw-telegram/Mattermost/__init__.py

from datetime import datetime

from telegram.utils.helpers import escape_markdown

import Helper
from Helper.MessageMD import MessageMD
```

```
class Mattermost(MessageMD):
    def __init__(self, payload: dict):
        self.channel_name = payload['channel_name']
        self.team_domain = payload['team_domain']
        self.text = payload['text']
        self.user_name = payload['user_name']
        self.trigger_word = payload['trigger_word']
        self.date = Helper.convert_utc_rome(
                datetime.fromtimestamp(payload['timestamp'] / 1000))
        super().__init__(self.gen_message())

    def gen_message(self):
        return escape_markdown(f'[{str(self.date.time())[:8]}] '
                               f'{self.user_name}@{self.channel_name} says:\n\n',
                                       version=2) + \
                f'```\n{escape_markdown(self.text, version=2)}```'
```

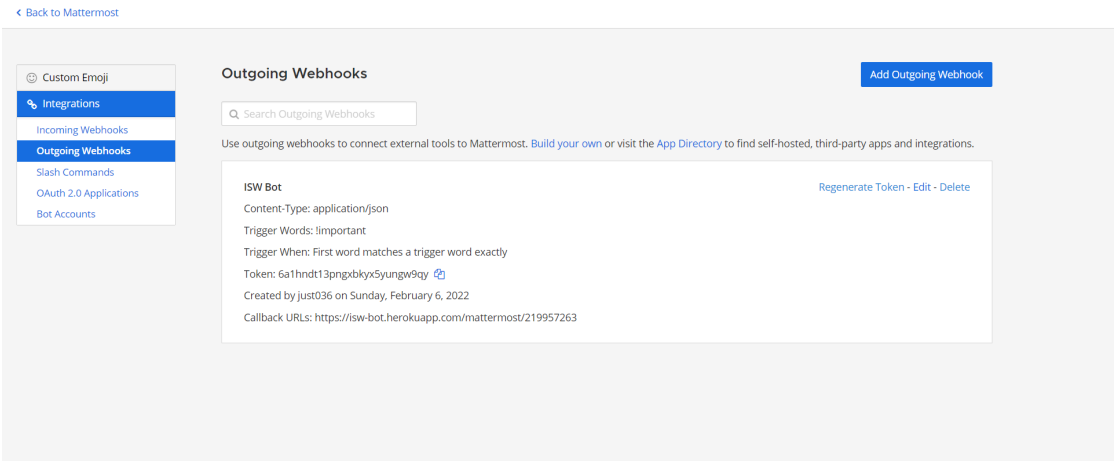

Figura 4.6: Webhook configurato nel sito di Mattermost

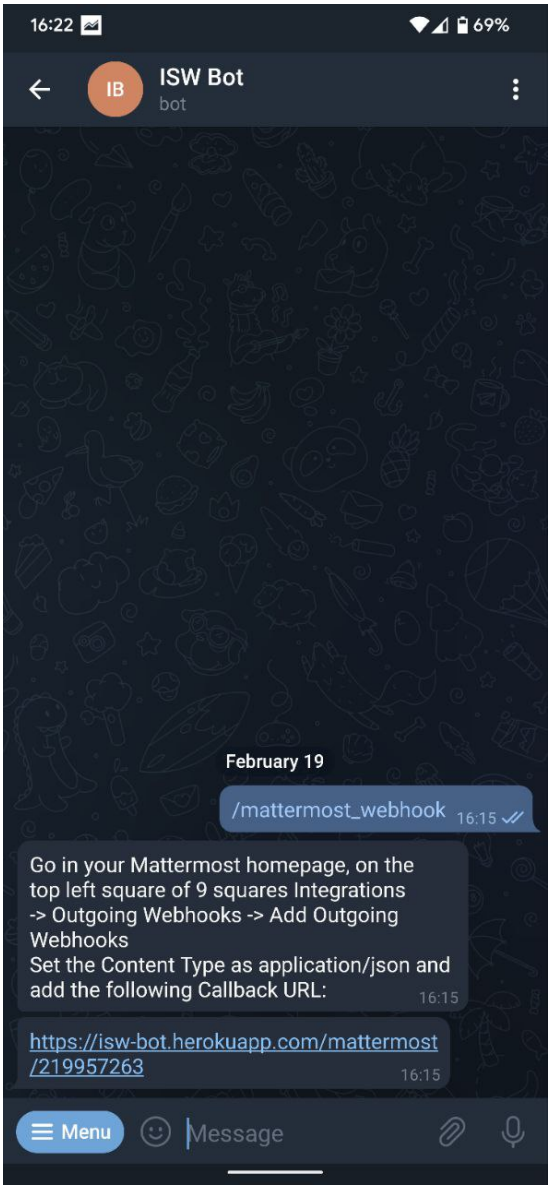

Figura 4.7: ISW Bot comando per configurare webhook di Mattermost

Figura 4.8: Mattermost esempio di echo ricevuto

4.2.4 Integrazione Taiga

Per integrare Taiga con il Bot, si deve invocare il comando /taiga_webhook che genera il link associato alla chat, andare nelle impostazioni della Board e inserire

l'URL prodotto nella sezione Webhooks di Integrations.

Secondo la documentazione di Taiga, vengono supportati quattro tipi di azioni e sei tipi di tipologie di eventi, i primi sono:

- create
- delete
- change
- test

I secondi, invece, sono:

- milestone
- userstory
- task
- issue
- wikipage
- test

Per ogni tipologia di evento si possono assumere diverse azioni. L'evento test è un caso separato, perché l'evento è associato solo ad un' unica azione, l'azione test.

Seguendo questo approccio, significa che la relazione tra eventi - azioni sono 1 a 3, in quanto i test sono 1 a 1 e possono essere esclusi. Per questo motivo, occorre innanzitutto determinare quale attività ha fatto scattare la chiamata POST al server e decidere di conseguenza cosa bisogna inviare.

L'idea più semplice è astrarre questa situazione in un albero di decisione nel seguente modo:

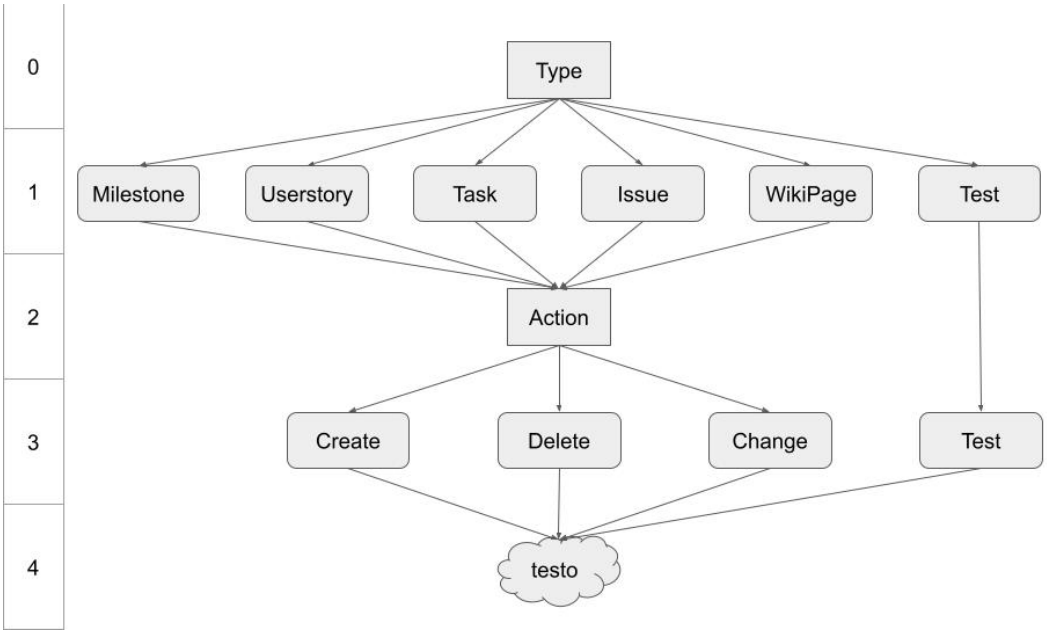

Figura 4.9: Albero di decisione per Taiga

Il livello 0 rappresenta il codice che inizia la scomposizione dell'oggetto JSON. Nei livelli 1-3 sono una serie di "filtri" e il livello 4 decide quale testo utilizzare ritornandolo al chiamante.

Più nello specifico, i testi vengono generati quasi a monte (al livello 1), tre per ognuno, da questo insieme di testi, al livello 4, verrà deciso quale utilizzare. Per determinare quale evento e azione ha fatto scaturire la chiamata, basterà usare su di essi uno statement per fare pattern matching.

Da notare che la gestione dell'arrivo di una richiesta da Taiga avviene a un livello ancora prima, che in questo caso è il chiamante.

Di seguito verrà mostrato il chiamante e l'astrazione dell'evento Milestone (livello 1-2), gli altri casi sono molto simili che differiscono per la più complessità della generazione dei messaggi, perchè serve che siano più specifici rispetto a Milestone:

```
# isw-telegram/app.py

@app.route(f"/taiga/<chat_id>", methods=["POST"])
```

```
def taiga_callback(chat_id):
    payload = json.loads(request.data)
    taiga = Taiga(payload)
    message, changes = taiga.format_message_md()
    try:
        if message is not None:
            isw_bot.send_message(chat_id=chat_id, text=message,
            parse_mode=PARSEMODE_MARKDOWN_V2)
        if changes is not None:
            isw_bot.send_message(chat_id=chat_id, text=changes)
        return "OK"
    except Exception as e:
        print(e)

# isw-telegram/Taiga/Type.py

class Milestone(MessageMD):
    def __init__(self, payload, action, changes=None):
        self.link = payload["permalink"]
        self.name = payload["name"]
        self.slug = payload["slug"]
        self.estimated_start = payload["estimated_start"]
        self.estimated_finish = payload["estimated_finish"]
        self.disponibility = payload["disponibility"]
        # questa linea di codice rappresenta il livello 2-3-4
        super().__init__(FormatAction(action,
                                      self._create_message_md,
                                      self._delete_message_md,
                                      self._change_message_md).message(),
                         get_changes(changes))

    @property
```

```
    def _create_message_md(self):
        return f"created {self._milestone_text_md}"

    @property
    def _delete_message_md(self):
        return f"deleted {self._milestone_text_md}"

    @property
    def _change_message_md(self):
        return f"changed {self._milestone_text_md}"

    @property
    def _milestone_text_md(self):
        return f"the milestone [{self.name}]({self.link})"
```

Il metodo **message()** della classe **FormatAction** astrae il livello 3-4 perchè deve decidere quale testo utilzzare:

```
# isw-telegram/Taiga/Type.py

class FormatAction:
    def __init__(self, action, create, delete, change):
        self.action = action
        self.create = create
        self.delete = delete
        self.change = change

    def message(self) -> str | None:
        match self.action:
            case Action.create:
                return self.create
            case Action.delete:
```

```
                return self.delete
            case Action.change:
                return self.change
            case _:
                return None
```

La superclasse **MessageMD** contiene un semplice campo chiamato **formatted_messages_md()** per ottenere il messaggio che è stato deciso:

isw-telegram/Helper/MessageMD.py

```
class MessageMD:
    def __init__(self, *formatted_messages_md: str):
        self.formatted_messages_md: tuple[str] = formatted_messages_md
```

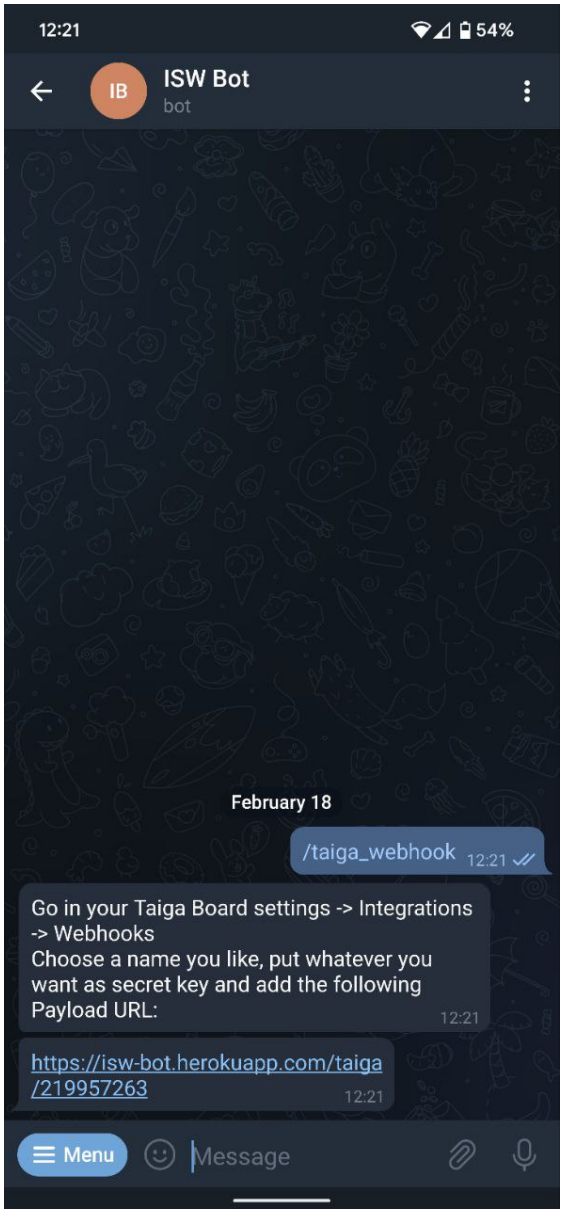

Figura 4.10: Generazione link per Taiga Webhook

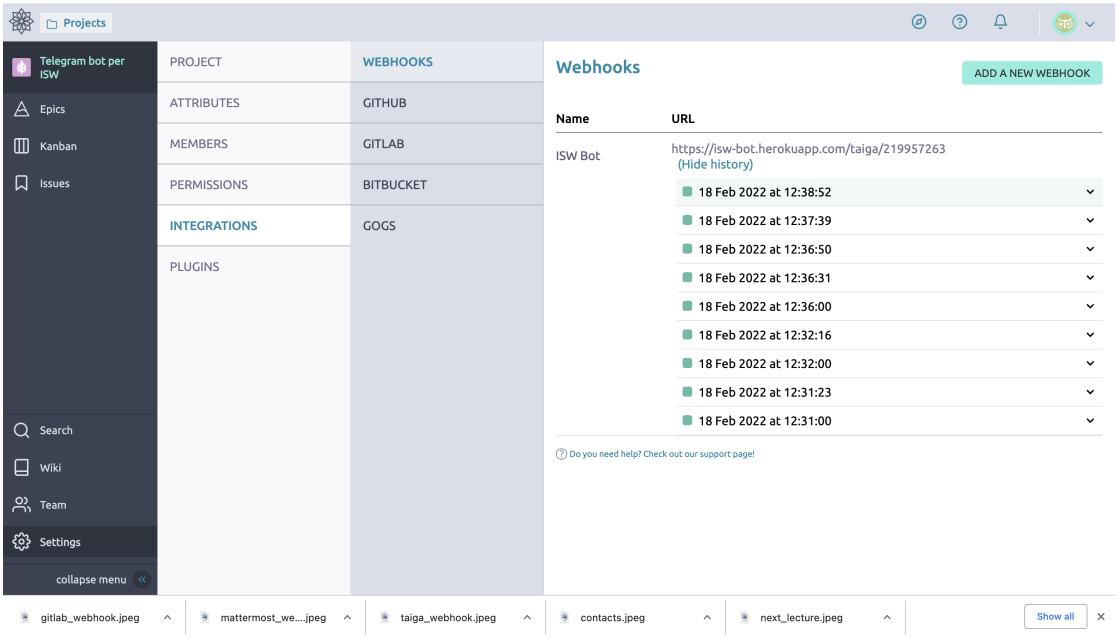

Figura 4.11: Integrazione di Taiga nella schermata della piattaforma

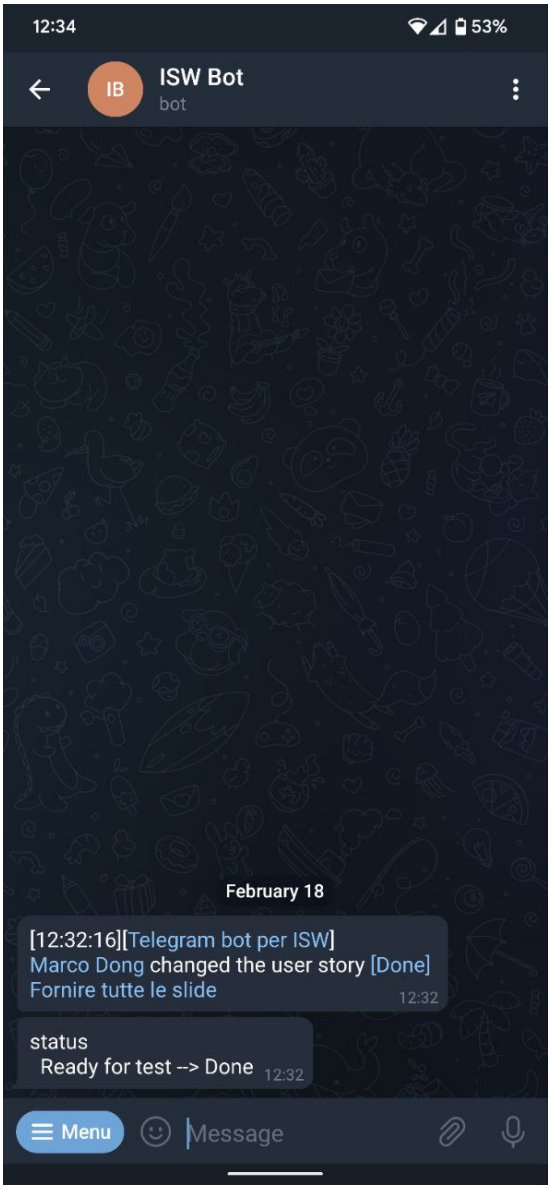

Figura 4.12: Esempio modifica di una userstory

Figura 4.13: Esempio di una modifica a una wiki

4.2.5 Integrazione Gitlab

Per integrare Gitlab con il Bot, il comando da invocare è /gitlab_webhook. Similmente, genera il un link collegato alla chat_id. Ottenuto quest'ultimo, si

deve recare alla sezione Settings del proprio progetto su Gitlab e aprire la pagina di Webhooks. Pure in questo caso il token segreto non è obbligatorio, per cui basta definire l'URL e la tipologia di eventi da sottoscrivere. Gli eventi supportati attualmente sono tre:

- Push events
- Issues events
- Wiki page events

E' presente una checkbox opzionale 'Enable SSL verification', abilitandola, il certificato SSL per le richieste HTTP in uscita viene verificato usando una lista interna di CA (Certificate Authority), questo significa che certificati 'self-signed' o che non appartengono alla lista non possono andar bene.

A differenza di Taiga, Gitlab fornisce oggetti più specifici e più dinamici. Per cui non è possibile seguire un approccio generico come fatto in precedenza con Taiga e Mattermost in quanto c'è da implementare un caso per ogni evento ed elaborare la complessità dei dati in modo tale da renderlo più leggibile alle persone.

Il seguente mostra l'albero di decisione per Gitlab:

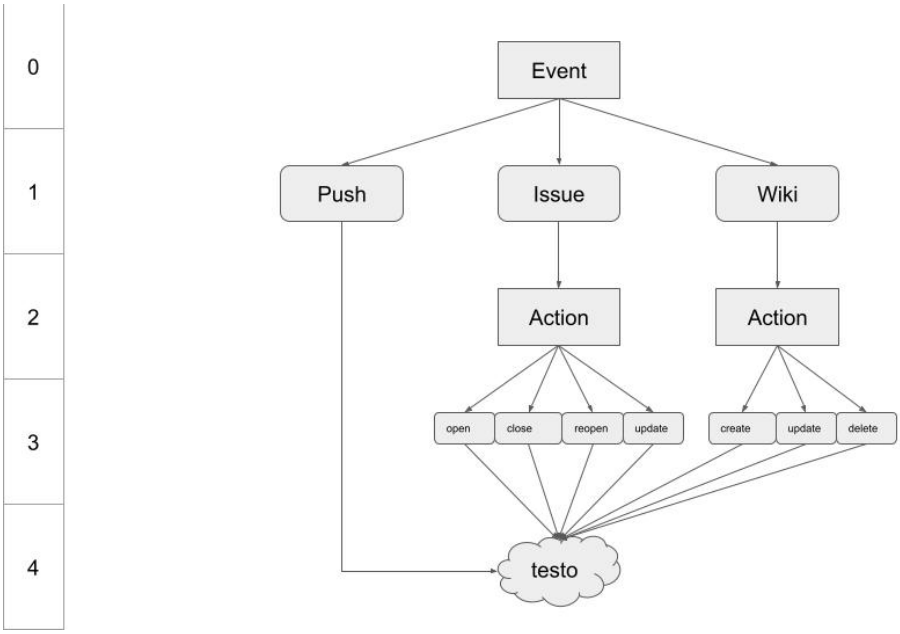

Figura 4.14: Albero di decisione per Gitlab

Di seguito il codice per il livello 0:

```
# isw-telegram/Gitlab/__init__.py

from Gitlab import Event
from Gitlab.Event import Push, Issue, Deployment, WikiPage


class Gitlab:
    def __init__(self, payload):
        self.payload = payload
        self.event_type = payload['object_kind']
        self.project_name = payload['project']['name']
        self.project_url = payload['project']['web_url']

    def get_event(self) -> Push | Issue | Deployment | WikiPage | None:
```

```
    # livello 0
        match self.event_type:
            case Event.push:
                return Push(self.payload)
            case Event.issue:
                return Issue(self.payload)
            case Event.deployment:
                return Deployment(self.payload)
            case Event.wiki_page:
                return WikiPage(self.payload)
            case _:
                return None
```

Di seguito, invece, verrà mostrato solo il codice per l'evento Push, in quanto la logica degli altri eventi è simile, nonostante cambi la struttura ricevuta e la complessità della generazione dei messaggi:

```
# isw-telegram/Gitlab/Event.py

from telegram.utils.helpers import escape_markdown

from Gitlab.Git import Commit, Repository
from Helper.MessageMD import MessageMD


class Push(MessageMD):
    def __init__(self, payload):
        self.branch = payload["ref"].split("/")[-1]
        self.last_commit_id = payload["after"]
        self.before_commit_id = payload["before"]
        self.user_name = payload["user_name"]
        self.user_username = payload["user_username"]
```

```
        self.user_profile_link = f"https://gitlab.com/{self.user_username}"
        self.repo = Repository(payload["project"])
        self.commits: list[Commit] = [Commit(commit) for commit in
                                                payload["commits"]]
        self.total_commits = payload["total_commits_count"]
        super().__init__(self.generate_message_md(),
                         self.generate_commits_md())

    def generate_message_md(self):
        user_name = escape_markdown(self.user_name, version=2)
        branch = escape_markdown(self.branch, version=2)
        repo_name = escape_markdown(self.repo.name, version=2)
        return f"[{user_name}]({self.user_profile_link}) just pushed
                                {self.total_commits} " \
               f"{'commit' if self.total_commits == 1 else 'commits'}" \
               f" to {branch} in [{repo_name}]({self.repo.url})\n"

    def generate_commits_md(self):
        text = "\n\n".join([f"\n{commit.formatted_message_md}"
                                        for commit in self.commits][::-1])
        if self.total_commits > len(self.commits):
            text += f"\n\nand *{self.total_commits - len(self.commits)}
                             more commits*"
        return text

# isw-telegram/Gitlab/Git.py

from datetime import datetime

from telegram.utils.helpers import escape_markdown
```

```
class Commit:
    def __init__(self, payload: dict):
        self.commit_id = payload['id']
        self.short_message = payload["message"].split("\n")[0].strip()
        self.time: datetime = datetime.fromisoformat(payload['timestamp'])
        self.url = payload['url']
        self.author_name = payload['author']['name']
        self.author_email = payload['author']['email']
        self.added: list[str] = payload['added']
        self.modified: list[str] = payload['modified']
        self.removed: list[str] = payload['removed']

    @property
    def formatted_message_md(self):
        return f"{self._formatted_message_md}\n" \
               f"```{self.formatted_added_md_escaped}\n" \
               f"{self.formatted_modified_md_escaped}" \
               f"\n{self.formatted_removed_md_escaped}```"

    @staticmethod
    def get_short_id(commit_id: str) -> str:
        return commit_id[:7]

    @property
    def _formatted_message_md(self):
        mess = escape_markdown(self.short_message, version=2)
        return f'\\[[{self.get_short_id(self.commit_id)}]({self.url})\\] {mess}'

    def _get_formatted_changes(self, changes: list[str], prepend="") -> str:
        n_changes = 5
        text = "\n".join([f"  {prepend} {commit}" for commit in
```

```
                                changes[:n_changes]])
        if len(changes) > n_changes:
            text += f"\nand {len(changes) - n_changes} more"
        return escape_markdown(text, version=2)

    @property
    def formatted_added_md_escaped(self):
        return self._get_formatted_changes(self.added, prepend="[+]")

    @property
    def formatted_modified_md_escaped(self):
        return self._get_formatted_changes(self.modified, prepend="[~]")

    @property
    def formatted_removed_md_escaped(self):
        return self._get_formatted_changes(self.removed, prepend="[-]")


class Repository:
    def __init__(self, payload: dict):
        self.name = payload['name']
        self.url = payload['web_url']
        self.description = payload['description']
```

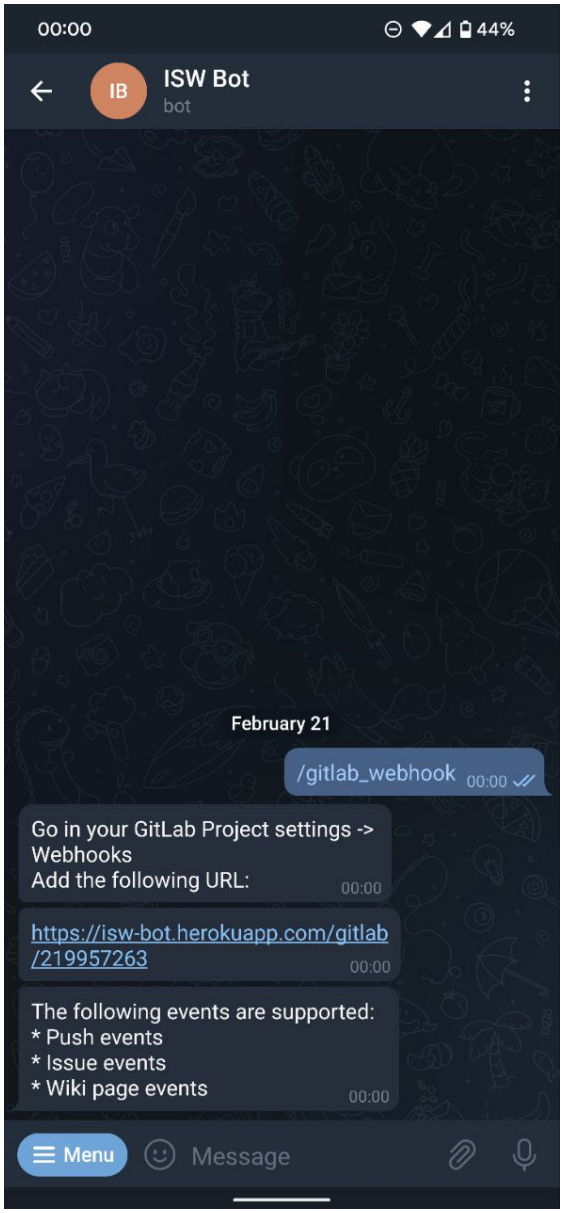

Figura 4.15: ISW Bot generazione link webhook

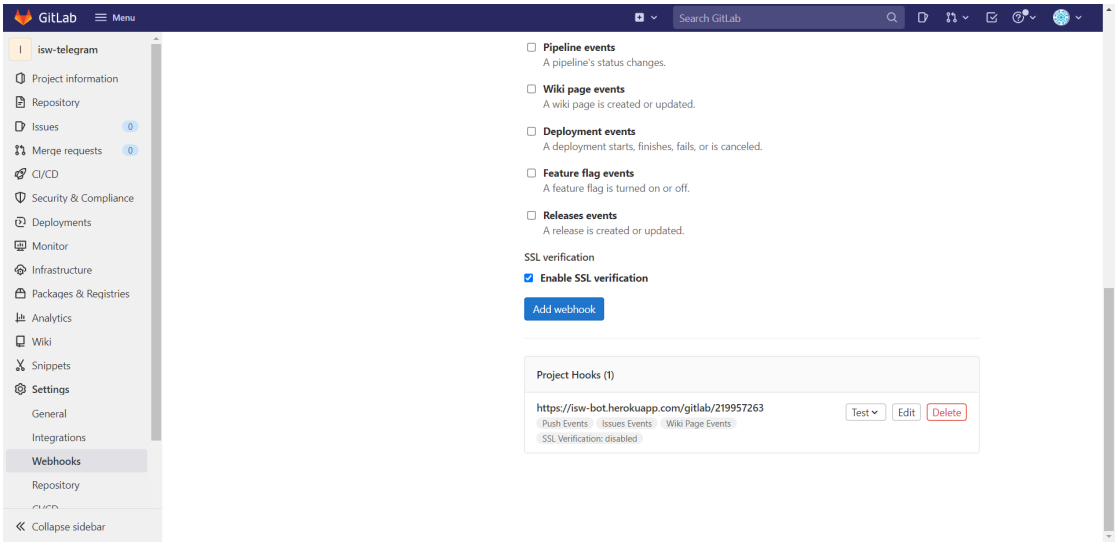

Figura 4.16: Gitlab webhook configurato

Evento Push

Ogni push alla repository di Gitlab fa scattare una chiamata POST. Anche se ci sono delle eccezioni a riguardo:

- push tags
- un push che influenza più di tre branch

Per ogni push, al massimo 20 oggetti commit sono inclusi all'interno del campo *commits*, i quali contengono principalmente le seguenti informazioni:

- *id* → **str** — hash del commit
- *message* → **str** — messaggio del commit
- *url* → **str** — link al commit
- *author* → **json** — le informazioni dell'utente che ha fatto il commit
- *added* → **array** — un vettore di percorsi dei file, partendo dalla cartella del progetto, che sono stati aggiunti

- *modified* → **array** — percorsi di quelli modificati
- *removed* → **array** — percorsi di quelli eliminati

ISW Bot appende all'inizio di ogni percorso del file una stringa che rappresenta la modifica fatta su di esso:

- `[+]` — simbolo 'più', sta per un nuovo file è stato aggiunto al progetto
- `[~]` — simbolo 'tilde', sta per un file è stato modificato
- `[-]` — simbolo 'meno', sta per un file è stato rimosso dal progetto

Inoltre il messaggio finale viene ordinato dal commit più recente, come su Gitlab.

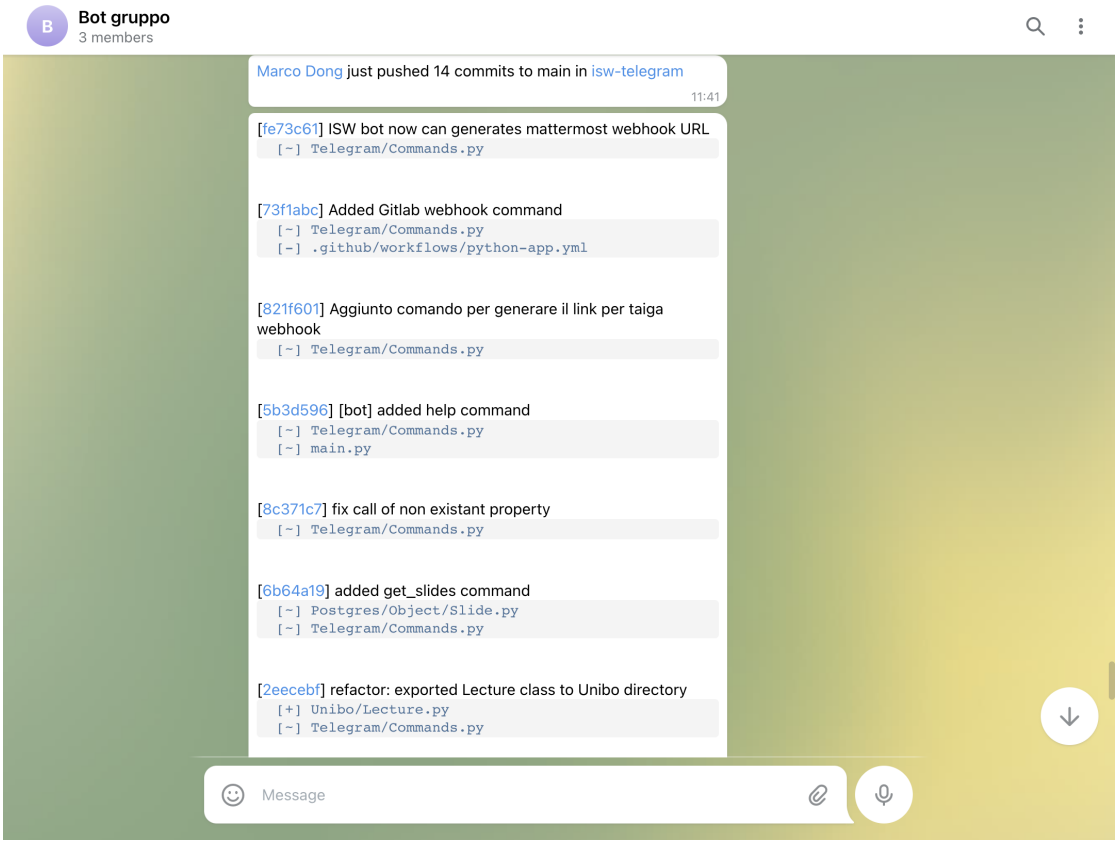

Figura 4.17: Gitlab push alla repository

Evento Issue

Come mostrato nella figura 4.14, l'evento Issue ha quattro tipi azioni:

- Open
- Close
- Reopen
- Update

I parametri che sono sempre presenti e che sono anche quelle più utili in merito a definire una Issue sono le seguenti:

- *object_attributes* → **JSON** — l'oggetto contiene una serie di informazioni relative al Issue stesso, come l'azione, la data di creazione, id, titolo, descrizione, url, stato (aperto o chiuso), label e cosi via...
- *repository* → **JSON** — l'oggetto che definisce il progetto che è associato al Issue
- *assignees* → **Array di JSON** — contengono informazioni sugli utenti che sono stati assegnati al Issue
- *changes* → **JSON** — è un oggetto composto in modo leggermente particolare ed è il campo più dinamico. Ogni modifica che viene fatta ha un campo composto dai campi '*previous*' e '*current*' , compreso alla creazione.

Esempio cambio di una label

```
"changes": {
    "updated_by_id": {
      "previous": ...,
      "current": ..
    },
    "updated_at": {
```

```
    "previous": "...",
    "current": ".."
  },
  "labels": {
    "previous": [{
      "id": 206,
      "title": "API",
      "color": "#ffffff",
      "project_id": 14,
      ...
    }],
    "current": [{
      "id": 205,
      "title": "Platform",
      "color": "#123123",
      "project_id": 14,
      ...
    }]
  }
}
```

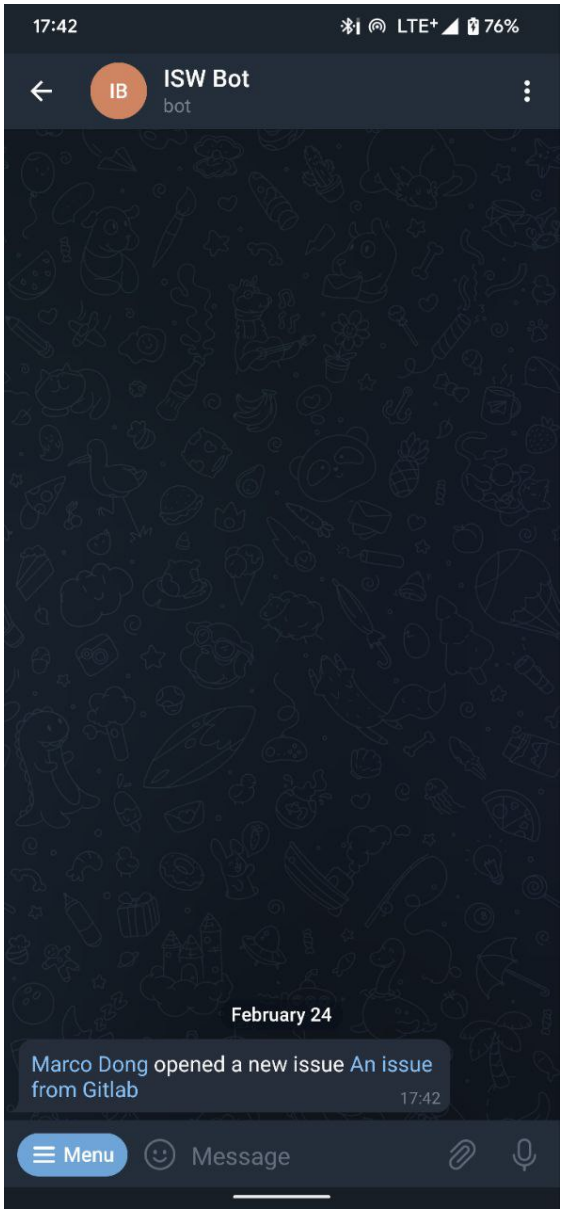

Figura 4.18: Creazione di una Issue su Gitlab

Evento Wikipage

Pure in questo caso possiamo riferirci alla figura 4.14, le azioni sono tre:

- create
- update
- delete

Le informazioni che servono per questa casistica sono contenute dentro il campo '*object_attributes*', che descrive il Wikipage.

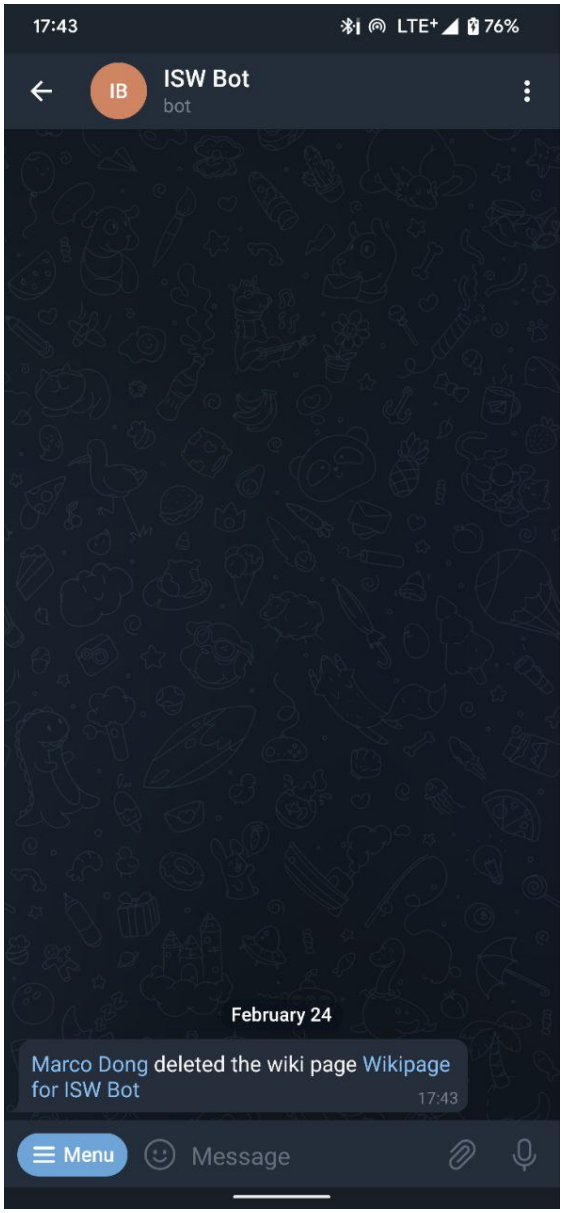

Figura 4.19: Eliminazione di una Wiki-page su Gitlab

Capitolo 5

Conclusione e sviluppi futuri

La comunicazione e la collaborazione sono fattori critici per sviluppare un software senza intoppi. Perciò esistono tool appositi in grado di mitigare o alleviare situazioni in cui non risulta facile metterle in atto.

Visto il vasto quantitativo di questi strumenti, risulta piuttosto comune che alcune persone all'interno di un team non vengano a conoscenza di certi cambiamenti avvenuti nel progetto. Per questo l'integrazione tra varie piattaforme all'interno di un Bot Telegram può essere una soluzione a questa problematica. Inoltre, rispetto ad altre applicazioni, Telegram è veloce, flessibile, sicuro e multi piattaforma. In aggiungenta a questo, è possibile considerarlo molto attraente perchè vi sono anche utenti 'normali' (o end user).

In questo progetto è stato quindi sviluppato un Bot in Telegram utilizzando il linguaggio di programmazione Python cercando di risolvere la problematica citata, attraverso l'uso di API di Telegram e Webhook da parte delle piattaforme di collaborazione.

Successivamente, si potrebbe ampliare il supporto a più piattaforme e anche integrarle in modo più profondo attraverso, non solo Webhook, ma anche API in modo da gestirle direttamente da ISW Bot. Ad esempio rispondendo da Telegram a un push fatto, si potrebbe far apparire quella risposta anche nella sezione dei commenti di quel commit su Gitlab, e rispondendo al commento, si commenta il commento!

Bibliografia

[1] Baden Eunson. “Team Communication”. In: gen. 2012. Cap. 18. ISBN: 9781742166179.

[2] Hans-Petter Halvorsen. *Python Programming*. 2019. ISBN: 978-82-691106-4-7.

[3] Kenneth A. Lambert. *Programmazione in Python*. Apogeo Education, 2013. ISBN: 978-8838786990.

[4] Nicolas Modrzyk. *Building Telegram Bots: Develop Bots in 12 Programming Languages Using the Telegram Bot API*. 1st. USA: Apress, 2018. ISBN: 1484241967.

[5] Paulo Silveira et al. “A Deep Dive into the Impact of COVID-19 on Software Development”. In: *IEEE Transactions on Software Engineering* (2021), pp. 1–1. DOI: 10.1109/TSE.2021.3088759.

Sitografia

[6] Gitlab. *Gitlab site*. https://about.gitlab.com/.

[7] Marco Dong. *Gitlab, Taiga, Mattermost webhook in Python*. 2022. URL: https://gitlab.com/User16421775608429967892/isw-telegram/-/tree/callback.

[8] Marco Dong. *ISW Bot in Python*. 2022. URL: https://gitlab.com/User16421775608429967892/isw-telegram/-/tree/main.

[9] Mattermost. *Mattermost*. https://mattermost.com/.

[10] Mattermost. *Mattermost Market Insight*. https://mattermost.com/wp-content/uploads/2021/10/451_Reprint_Mattermost_14OCT2021.pdf.

[11] McKinsey. *McKinsey Survey on shift to remote working*. https://www.mckinsey.com/business-functions/people-and-organizational-performance/our-insights/revisiting-agile-teams-after-an-abrupt-shift-to-remote. 2020.

[12] Python. *Python Documentation*. https://docs.python.org/.

[13] Taiga. *Taiga Site*. https://taiga.io/.

[14] Telegram. *Python Telegram Bot Documentation*. https://python-telegram-bot.readthedocs.io/.

[15] Telegram. *Python Telegram Bot module github*. https://github.com/python-telegram-bot/python-telegram-bot.

[16] Telegram. *Telegram bot APIs*. https://core.telegram.org/bots/api.

[17] Wikipedia contributors. *Python (programming language) — Wikipedia, The Free Encyclopedia*. [Online; accessed 13-February-2022]. 2022. URL: https://en.wikipedia.org/w/index.php?title=Python_(programming_language)&oldid=1071456939.

[18] Wikipedia contributors. *Telegram (software) — Wikipedia, The Free Encyclopedia*. [Online; accessed 6-February-2022]. 2022. URL: https://en.wikipedia.org/w/index.php?title=Telegram_(software)&oldid=1069143970.

Appendice A

Guida all'uso

Le tre piattaforme che possono essere utilizzate col Bot sono Mattermost, Taiga e Gitlab. Queste tre integrazioni sono indipendente una dall'altra, per cui si possono utilizzare tutte, una solamente o nessuna. In questo capitolo di appendice verrà spiegato passo-passo in modo più esaustivo e meno tecnico su come poter iniziare e quali sono i requisiti per poter usare il prodotto di questo progetto.

Innanzitutto occorre avere un account sulle seguenti piattaforme, se si vuole utilizzare uno qualsiasi di questi:

- Telegram — https://web.telegram.org/z/
- Taiga — https://tree.taiga.io/discover/
- Gitlab — https://about.gitlab.com/
- Mattermost — https://mattermost.com/

In tutte le piattaforme non ci sono processi di creazione di account particolarmente costosi in termini di tempo e complicati. Inoltre è possibile utilizzarli anche solamente nella versione web, evitando di dover scaricare programmi e aggiungere ulteriore overhead a queste operazioni.

Una nota importante è che Mattermost è gratuita solo se viene implementato su un server proprio, altrimenti è possibile usufruire della prova gratuita di 14 giorni semplicemente fornendo e confermando la propria email.

Una volta che si ha un account su quelle piattaforme si può iniziare a interagire con il Bot. Si anticipa che verrà usato uno smartphone Android con l'applicazione Telegram installata dal Play Store e la versione web per le altre tre, perciò potrebbero variare leggermente alcuni passaggi, ad esempio per la versione dell'applicazione o differenze tra piattaforme (app desktop o su smartphone).

A.1 Telegram

Il bot si chiama `@isw_cs_unibo_bot`, trovabile tramite la ricerca globale di Telegram. Per iniziare si dovrà cliccare sul bottone "Start" e poi invocare i comandi desiderati. Il bot può essere usato sia tramite una chat privata che aggiungerlo in un gruppo nuovo o già esistente, il funzionamento di esso non ha differenza.

A.1.1 Chat privata

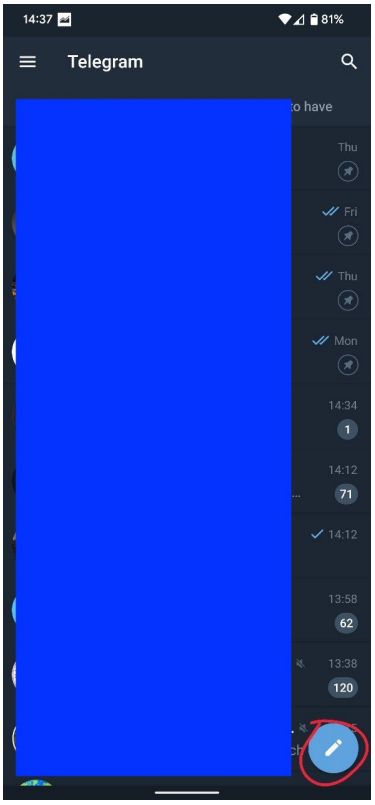

Figura A.1: Creare una nuova chat

Figura A.2: Ricerca globale per ISW Bot

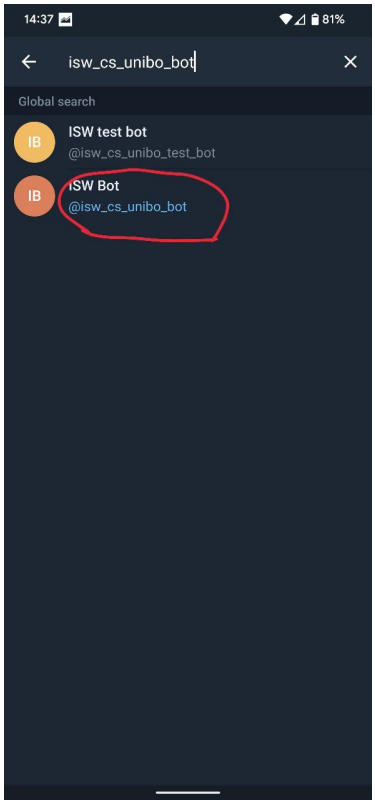

Figura A.3: Selezionare ISW Bot

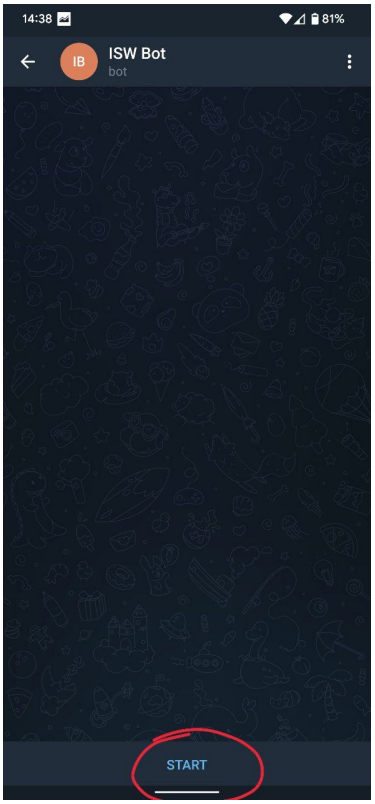

Figura A.4: La chat di ISW il bottone start

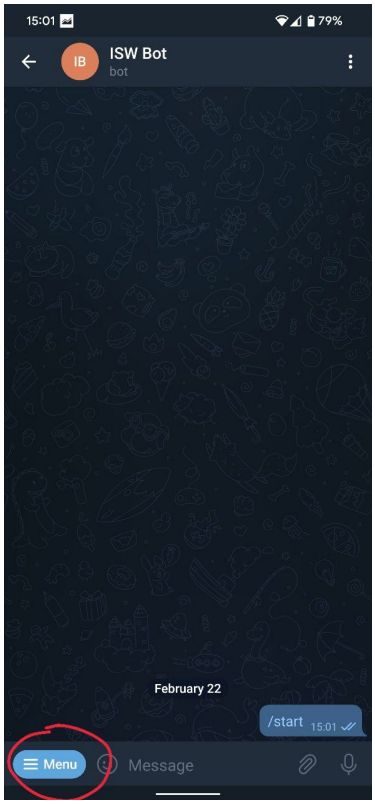

Figura A.5: Aprire il menu di ISW Bot

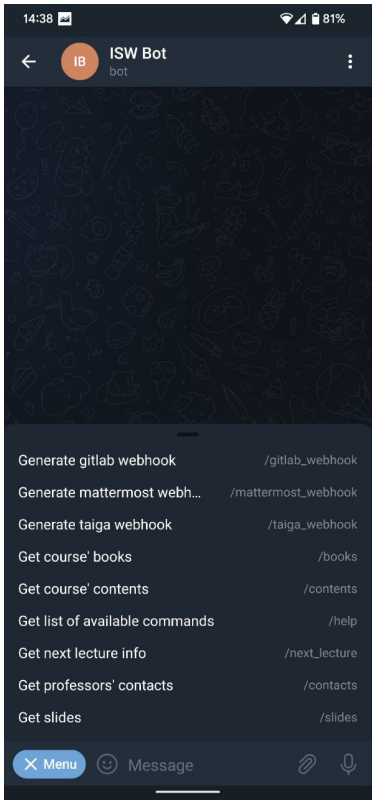

Figura A.6: Lista di comandi

A.1.2 Creazione di nuovo gruppo

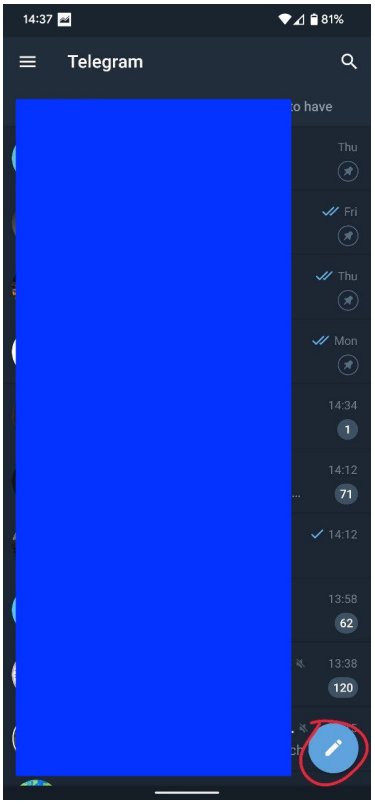

Figura A.7: Creare una nuova chat

Figura A.8: Creazione di un gruppo

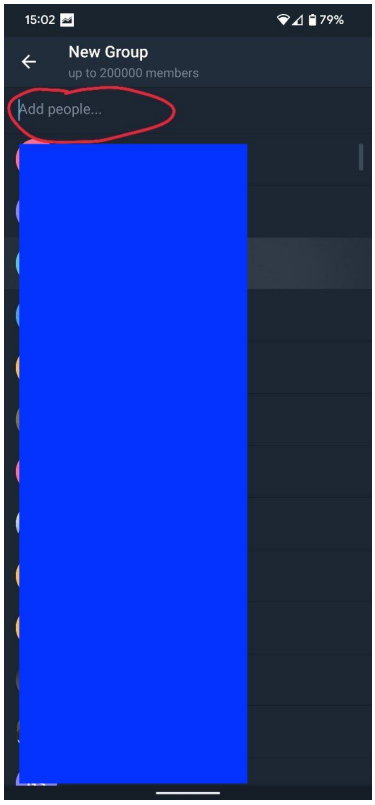

Figura A.9: Aggiunta membro del gruppo

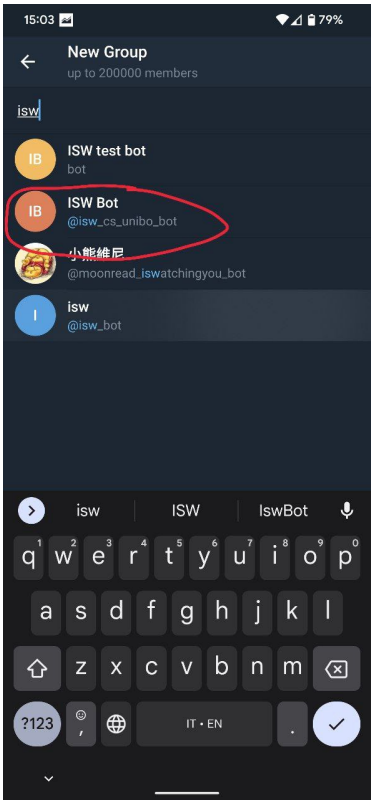

Figura A.10: Ricerca globale per ISW Bot durante la creazione di un gruppo

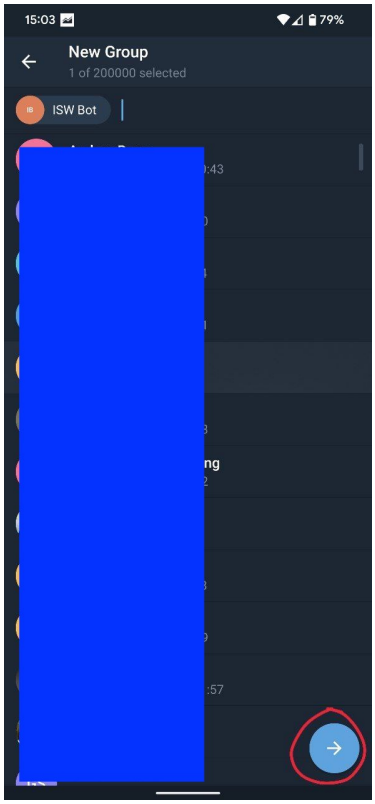

Figura A.11: Conferma dell'aggiunta dei nuovi membri

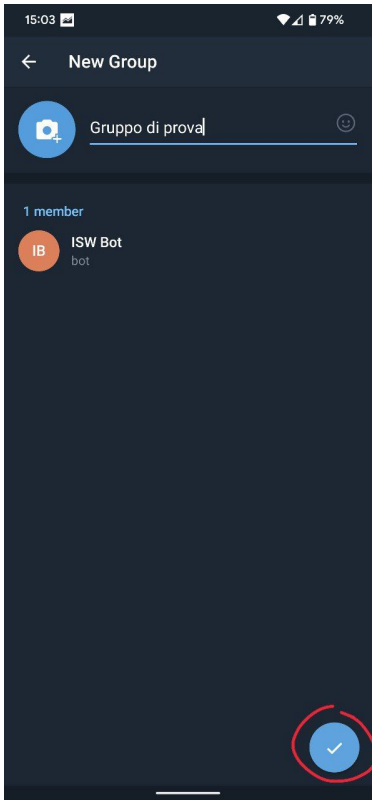

Figura A.12: Conferma della creazione di un nuovo gruppo

Se il gruppo esiste già

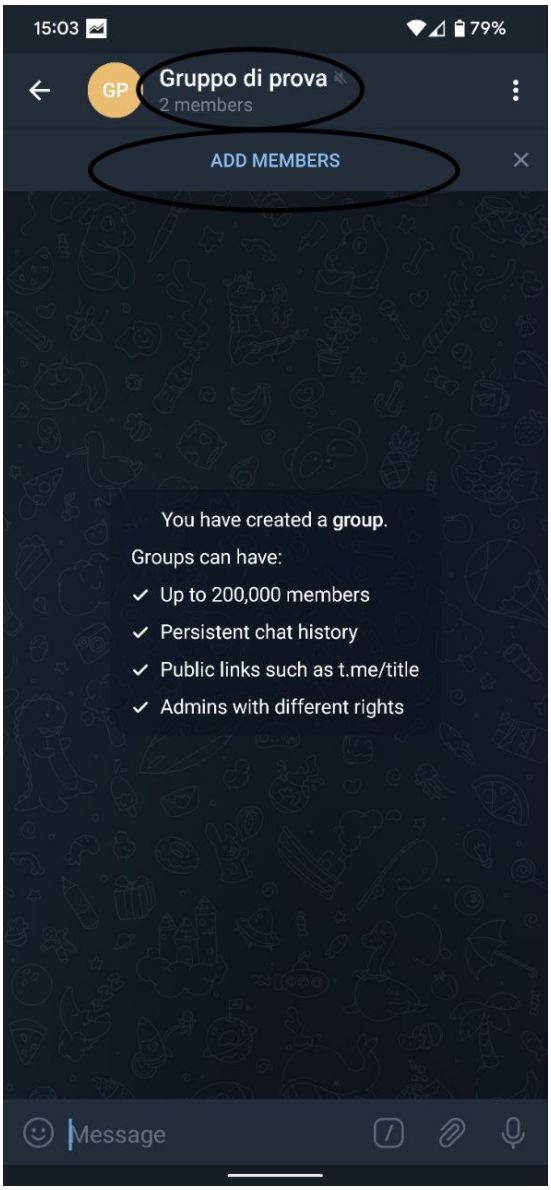

Figura A.13: Cliccare su uno dei due all'interno di un gruppo, quello più in basso a volte non c'è

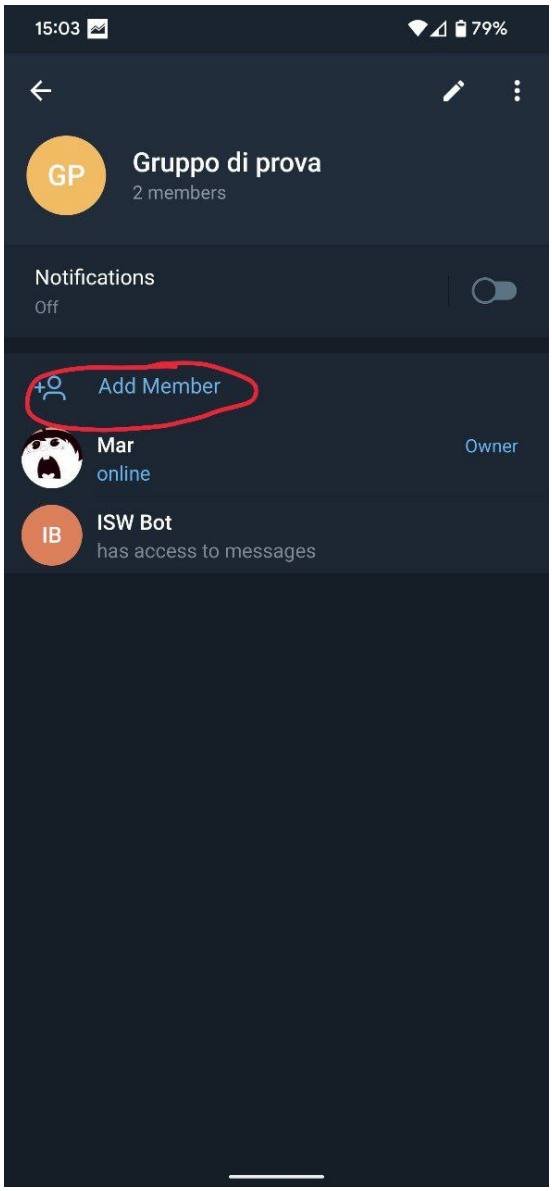

Figura A.14: Aggiunta di un nuovo membro tramite le impostazioni del gruppo

Aprire la lista di comandi

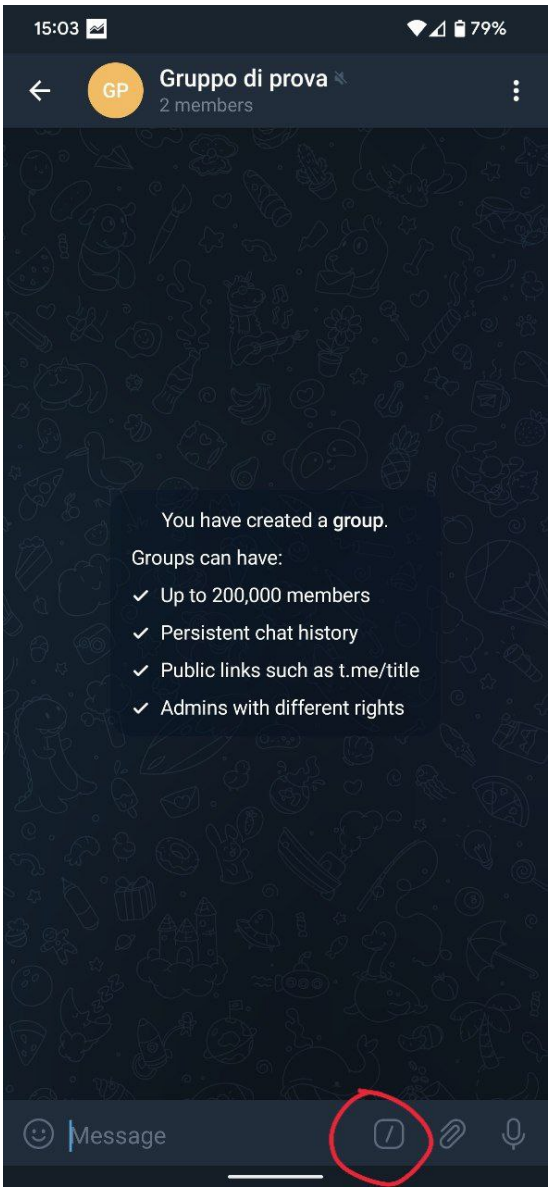

Figura A.15: Il bottone menu

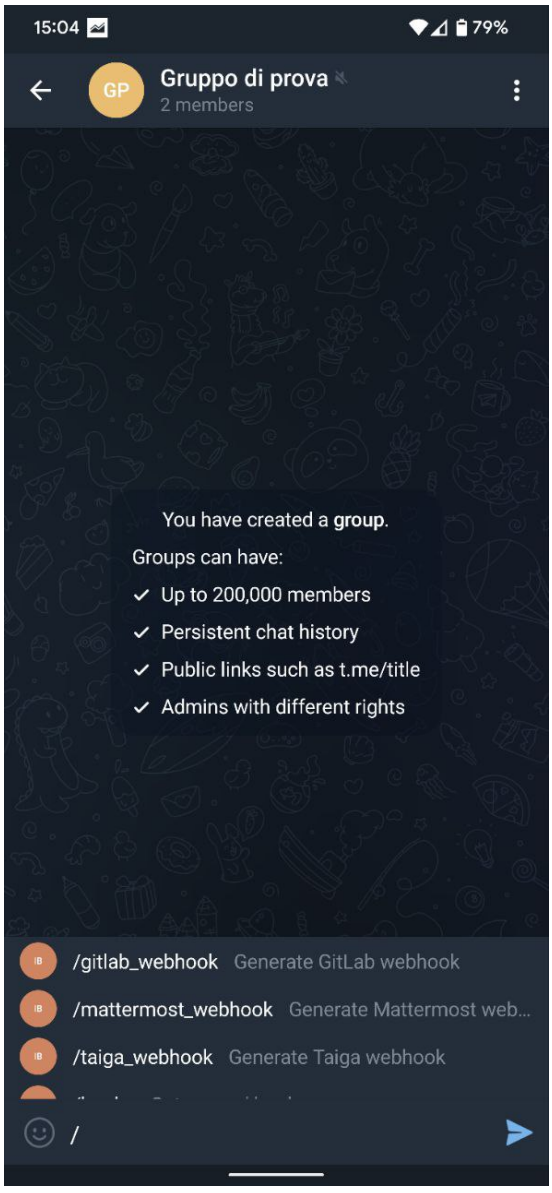

Figura A.16: La lista di comandi

A.2 Mattermost

Viene configurato come si ritiene opportuno, la chiave segreta non ha alcuna importanza, ma il 'content-type' deve essere di tipo 'application/json'.

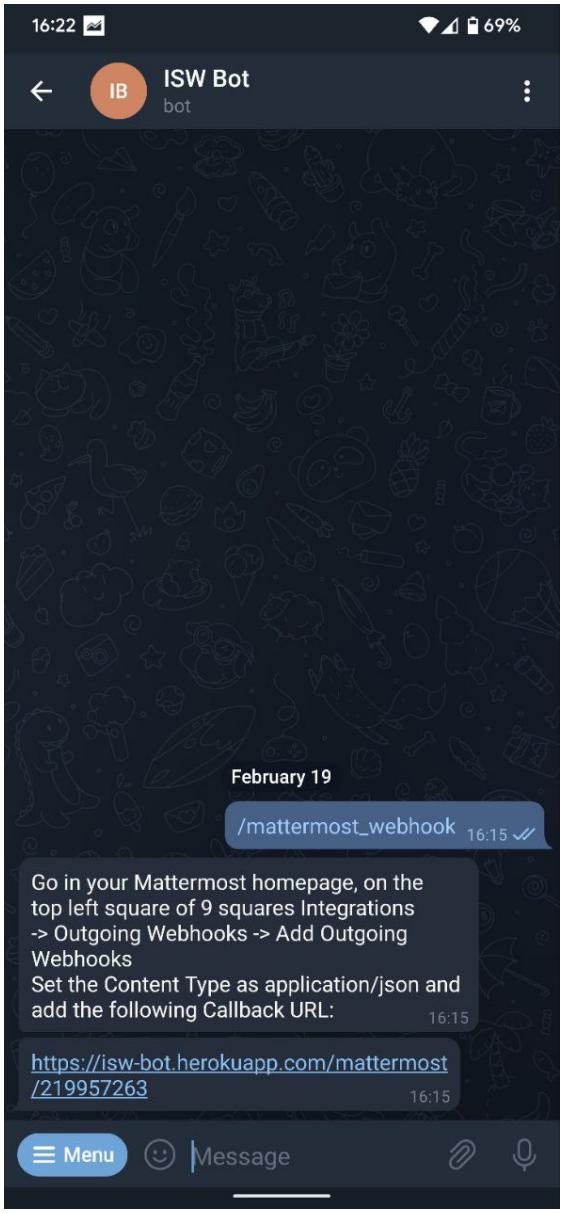

Figura A.17: Generazione link mattermost

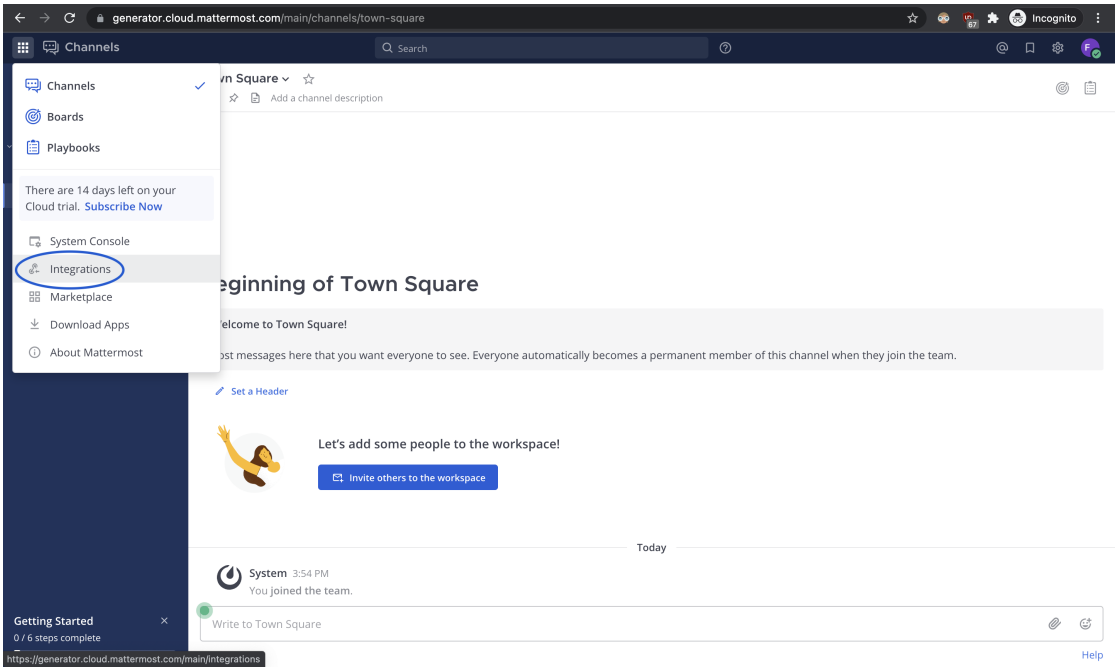

Figura A.18: Homepage di Mattermost

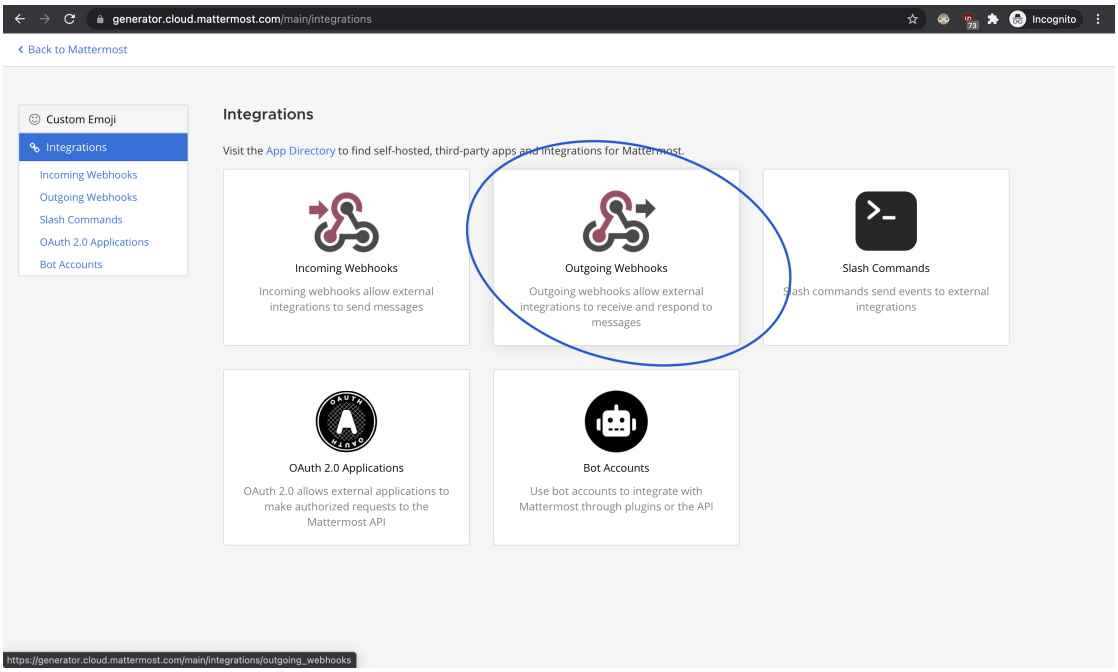

Figura A.19: Outgoing Webhooks

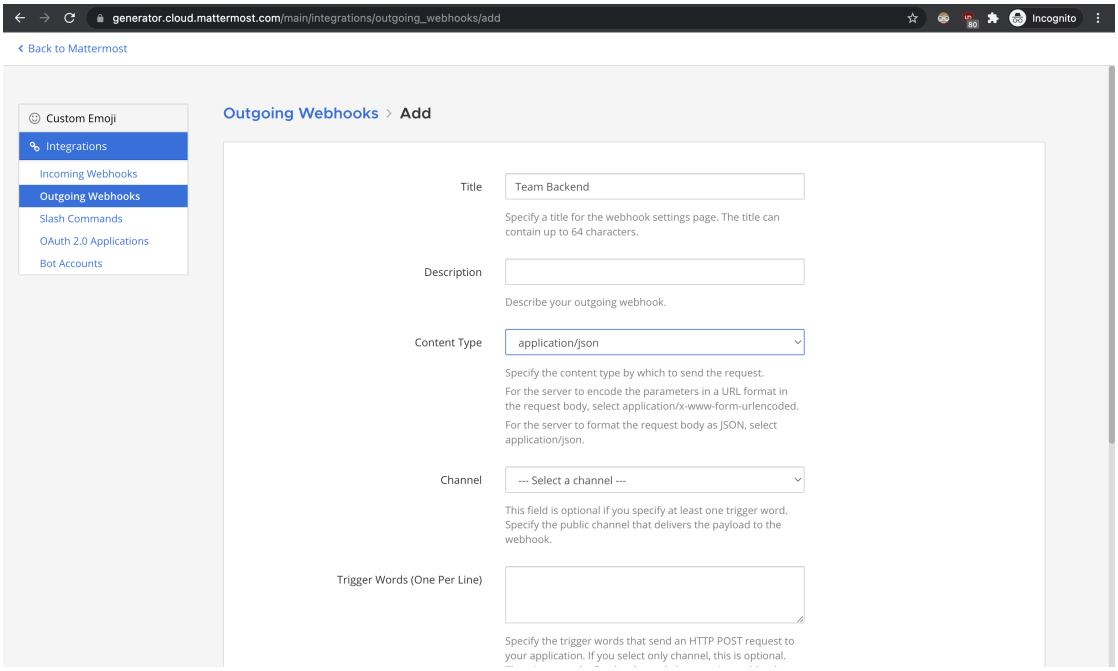

Figura A.20: Scherma di configurazione Webhook

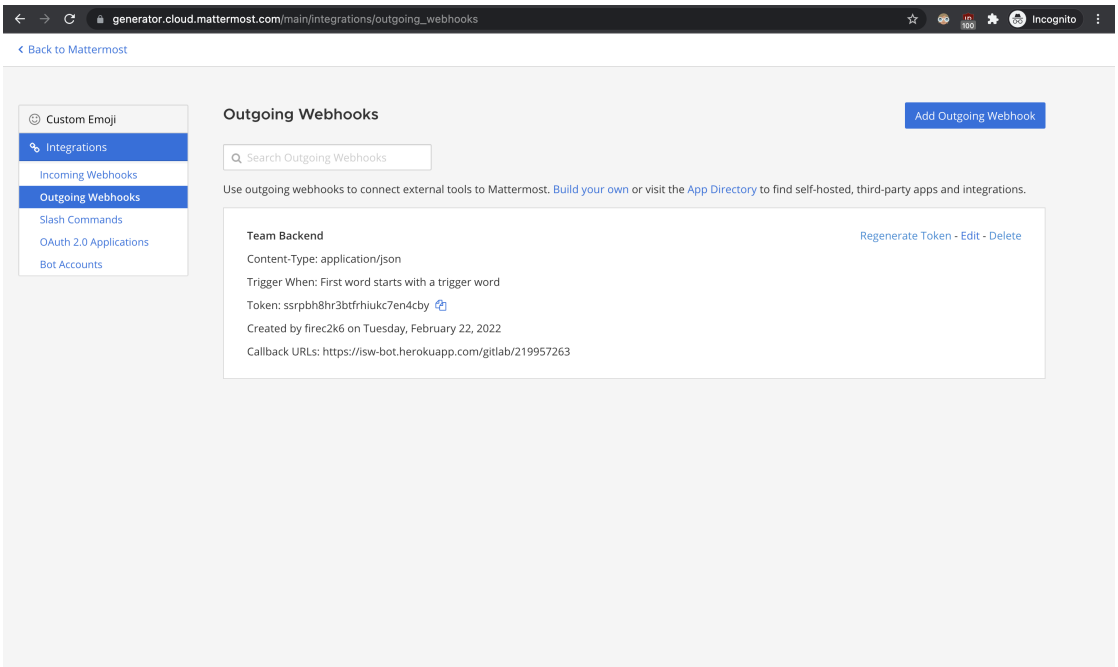

Figura A.21: Esempio di Webhook configurato

A.3 Taiga

La configurazione di Taiga impone che la chiave segreta sia definita, ma siccome ISW Bot non fa nessun controllo, può essere inserito come chiave una qualsiasi stringa.

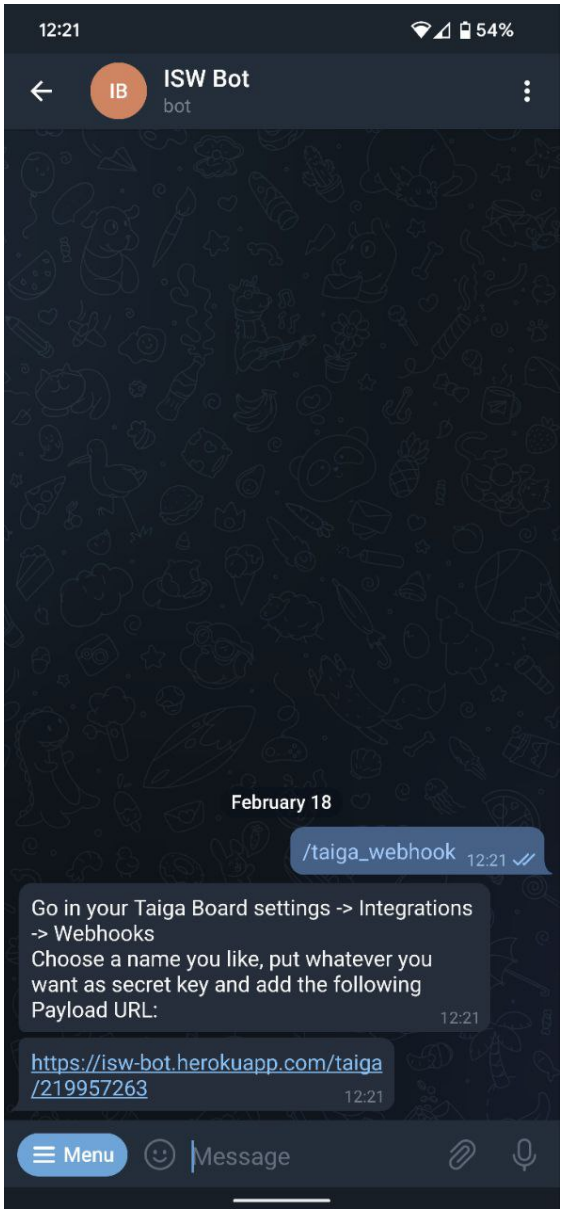

Figura A.22: Generazione link per Taiga

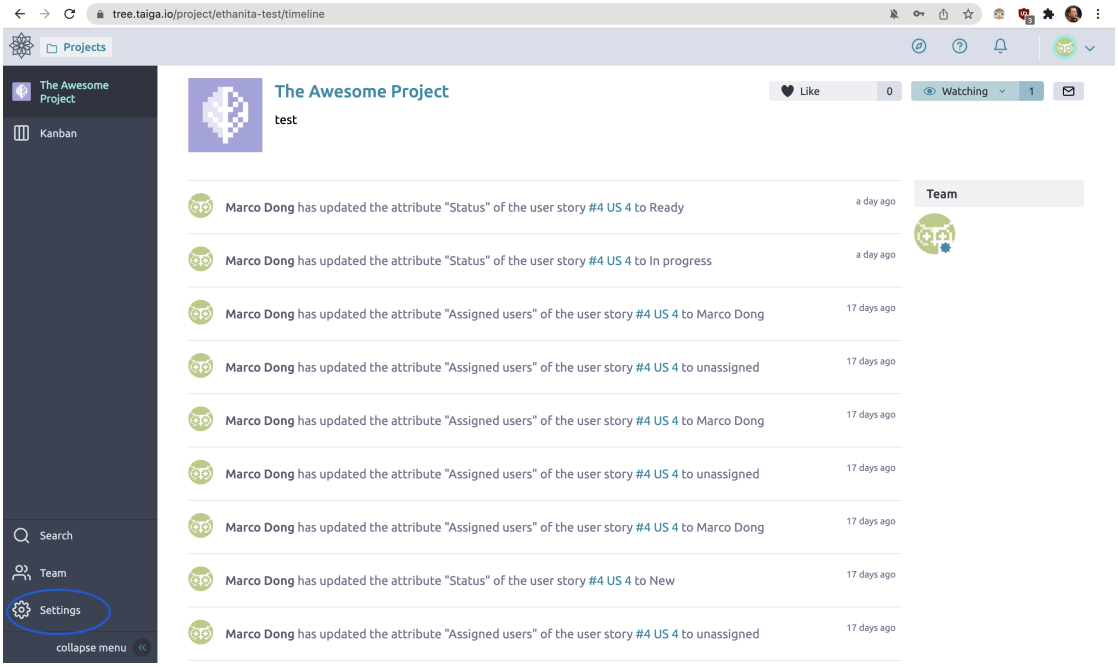

Figura A.23: Homepage di Taiga

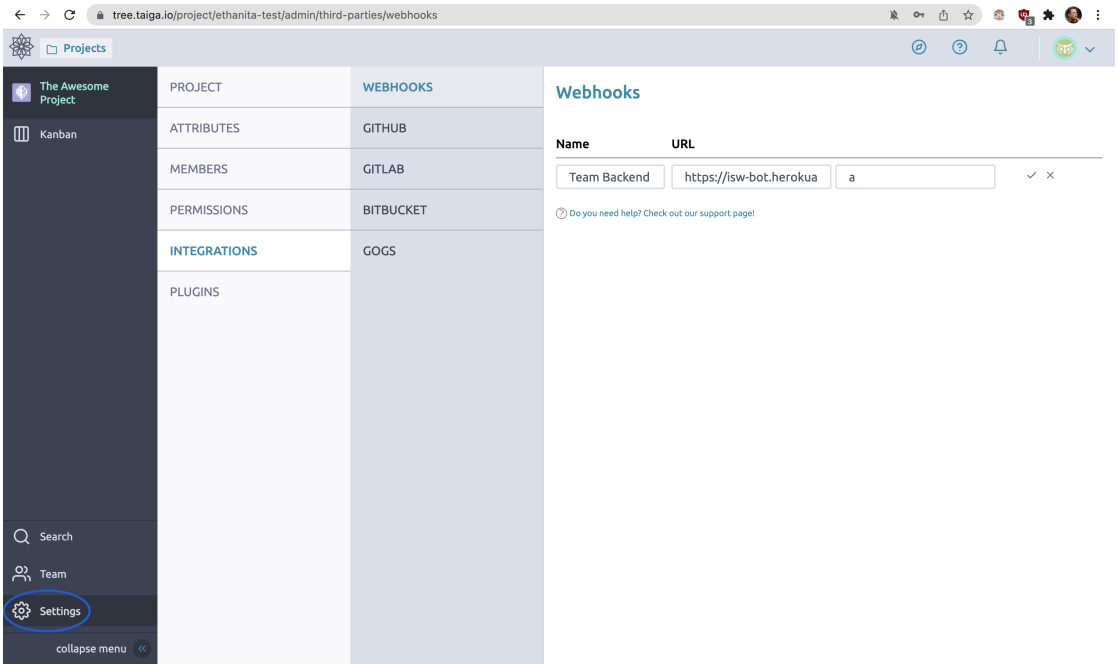

Figura A.24: Creazione di un nuovo webhook

A.4 Gitlab

Nella configurazione del webhook di Gitlab, solo tre tipi di eventi sono attualmente supportati da ISW Bot:

- Push
- Issue
- Wiki

Inoltre si consiglia di disattivare il flag che abilita la verifica SSL, in quanto non porta nessun beneficio nel nostro contesto, per di più potrebbe portare problemi come l'impossibilità di ricevere le richieste da Gitlab.

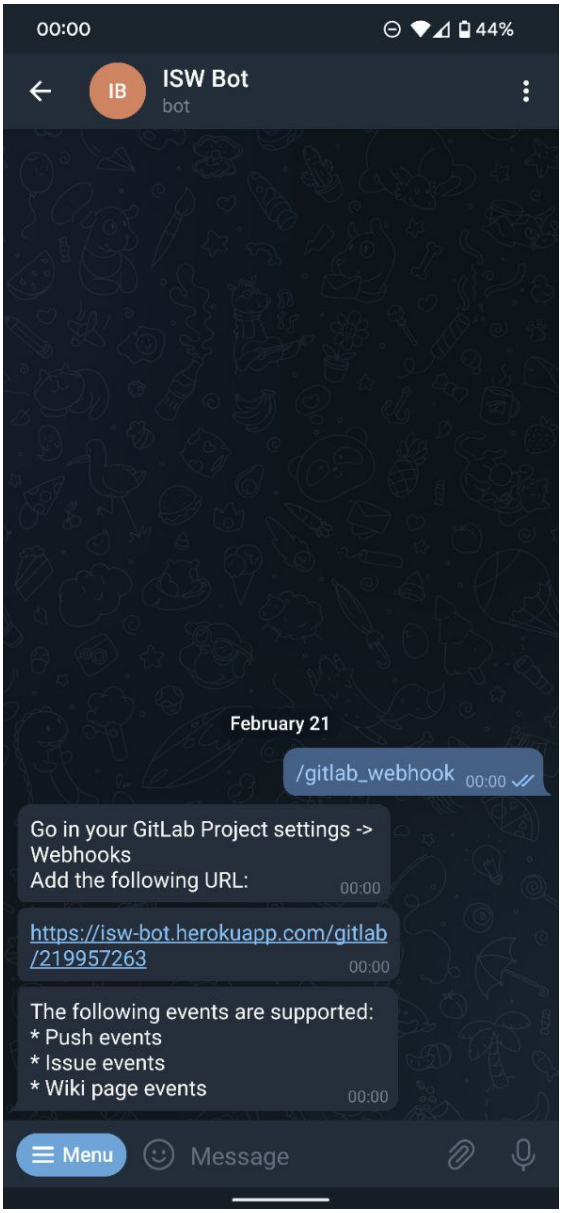

Figura A.25: Generazione link per Gitlab

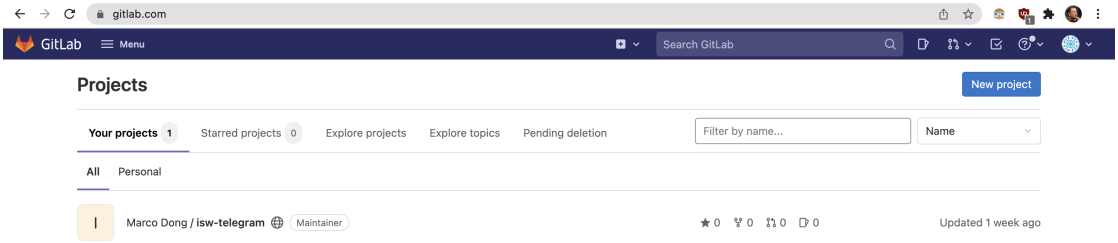

Figura A.26: Homepage di Gitlab, lista di progetti

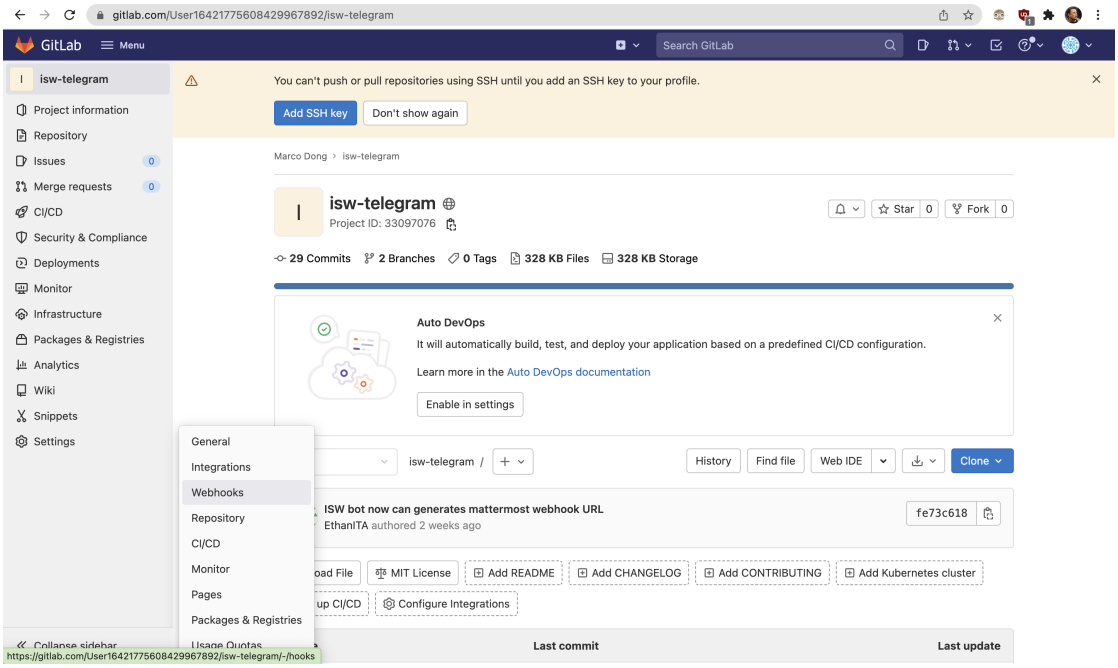

Figura A.27: Homepage del progetto, Settings -¿ Webhooks

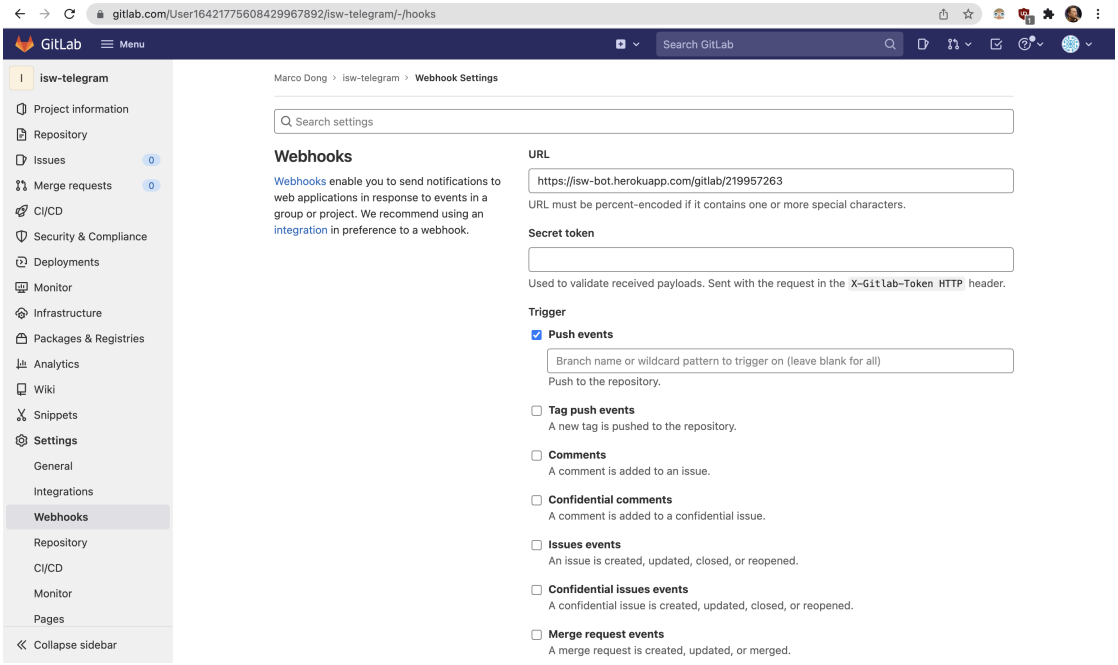

Figura A.28: Configurazione del webhook, selezione degli eventi

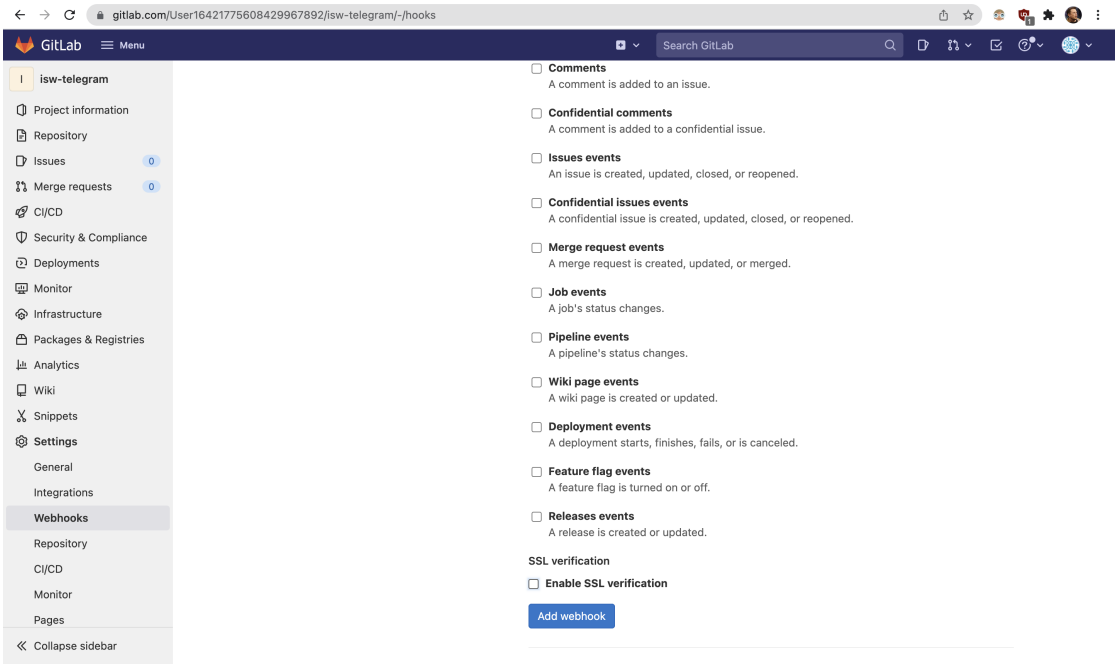

Figura A.29: Aggiunta del webhook