T.O.A.S.T

Un bot Discord per l'ispezione manuale dei Community Smells

Corso: Ingegneria Gestione ed Evoluzione del Software

Prof. Andrea De Lucia
Dott. Antonio Della Porta

Università degli Studi di Salerno
Anno accademico: 2023-2024

Repository GitHub:
https://github.com/Ferdi00/toast-tool

Team Members
Ferdinando Boccia
f.boccia28@studenti.unisa.it
Domenico D'Antuono
d.dantuono7@studenti.unisa.it

Contents

1 Proposte di modifica e contributi

1.1 CR_1: Creazione di API per i servizi di TOAST.

Descrizione
Creazione di API per i servizi di TOAST.
Motivazione
La creazione di API per i servizi offerti dall'applicazione TOAST (aggiunta e analisi dei collaboratori), permettono di disaccoppiare il tool da Discord, rendendo tali servizi accessibili non solo tramite l'applicazione, ma anche attraverso altre piattaforme e servizi esterni.
Priorità
alta
Effort
alto
Conseguenze se non accettata
I servizi continueranno ad essere utilizzabili soltanto tramite il bot discord.
Da fare dopo
NA

Table 1: CR_1: Creazione di API per i servizi di TOAST.

1.1.1 Metodologia

Attraverso un processo di reverse engineering, verranno individuate le funzionalità specifiche dei vari moduli, concentrandosi particolarmente su quelli responsabili dell'I/O, dell'analisi dei dati e dell'aggiunta dei collaboratori. Una volta completata questa fase di analisi, si procederà con lo sviluppo della componente relativa al web-service. Questa parte del progetto sarà realizzata mediante la creazione di Rest API utilizzando Express.js. Express.js è un framework JavaScript che offre un insieme completo di funzionalità per la creazione di applicazioni web e mobile, facilitando lo sviluppo di servizi HTTP RESTful.

1.1.2 Risultati attesi

Se la change request fosse approvata, il tool beneficerebbe di diverse migliorie, tra cui:

- **Disaccoppiamento dei servizi**: Creare un'API separata consente di disaccoppiare il servizi da Discord, rendendo il sistema più modulare e scalabile. Questo significa che è possibile aggiungere o modificare funzionalità nell'API senza dover necessariamente modificare il bot Discord.
- **Accessibilità esterna**: L'API consente l'accesso alle funzionalità di analisi dei dati e aggiunta di collaboratori da parte di app e servizi esterni. Ciò significa che è possibile utilizzare i risultati dell'analisi anche in contesti al di fuori di Discord, come ad esempio in applicazioni web, mobile o in altri servizi di terze parti.
- **Facilità di manutenzione**: Separare le funzionalità di analisi dei dati e aggiunta di collaboratori in un'API permette una gestione più agevole e una manutenzione più semplice. È più facile identificare e risolvere eventuali problemi o aggiornare le funzionalità senza influenzare il funzionamento del bot Discord.
- **Scalabilità**: Separando le funzionalità di analisi dei dati in un'API, è più facile scalare il sistema in modo indipendente. Ad esempio, è possibile distribuire l'API su più server o utilizzare servizi di cloud computing per gestire carichi di lavoro elevati senza influenzare le prestazioni del bot Discord.

1.2 CR_2: Sviluppo di un'applicazione web con interfaccia grafica migliorata.

Descrizione
Sviluppo di un'applicazione web con interfaccia grafica migliorata che utilizza le API del tool
Motivazione
Attualmente il tool può essere eseguito unicamente via Discord, inoltre sono necessari una serie di passaggi per il setup e la configurazione. Per questo motivo si è pensato di sviluppare anche una web app. L'applicazione web sfrutterebbe i servizi di TOAST attraverso l'utilizzo delle API, inoltre ci si vuole concentrare sullo sviluppo di un'interfaccia grafica moderna pensata appositamente per l'utilizzo del tool in maniera più semplice e diretta e allo stesso tempo che possa migliorare l'esperienza utente.
Priorità
alta
Effort
alto
Conseguenze se non accettata
Il tool potrà essere utilizzato solo tramite discord o applicazioni di terze parti (se qualcuno deciderà di utilizzare le API).
Da fare dopo
CR_1

Table 2: CR_2: Sviluppo di un'applicazione web con interfaccia grafica migliorata.

1.2.1 Metodologia

L'obiettivo primario consiste nello sviluppare un'applicazione web operativa sfruttando le più recenti tecnologie disponibili. Per quanto concerne il front-end, al fine di mantenere coerenza con l'ambiente JavaScript, si prevede l'impiego di un framework avanzato come React.js o Next.js. In aggiunta, si prevede l'utilizzo di

librerie esterne open source, mirate ad arricchire l'esperienza utente attraverso un miglioramento della parte grafica e delle animazioni.

Per quanto riguarda il backend, si adotteranno Express.js e Node.js, al fine di semplificare e accelerare il processo di sviluppo. Inoltre, si intendono impiegare i web service precedentemente sviluppati per condurre analisi dei dati, consentendo così un'integrazione sinergica e ottimizzata tra le diverse componenti dell'applicazione. Tale approccio favorirà una maggiore efficienza nell'erogazione dei servizi offerti dall'applicazione web, garantendo al contempo un'esperienza utente più fluida e completa.

1.2.2 Risultati attesi

Se la change request fosse approvata, il tool beneficerebbe di diverse migliorie, tra cui:

- **Controllo completo dell'esperienza utente**: Con un'app dedicata, si ottiene il pieno controllo sull'interfaccia utente e sull'esperienza dell'utente, consentendo una personalizzazione più approfondita in linea con le esigenze specifiche degli utenti, migliorando così l'usabilità e l'accessibilità di TOAST.
- **Accessibilità globale**: L'applicazione sarà accessibile da qualsiasi luogo e dispositivo connesso a Internet. Questo consente agli utenti di utilizzare il servizio senza dover installare o configurare nulla localmente.

1.3 CR_3: Deploy dell'app su una piattaforma di hosting online.

Descrizione
Deploy dell'app su una piattaforma di hosting online.
Motivazione
Il tool attualmente per essere utilizzato necessita di essere avviato in locale. Per questo motivo, si valuta la possibilità di pubblicare l'applicazione ed i servizi su una piattaforma di hosting online, in modo da garantire la disponibilità continua del servizio.
Priorità
alta
Effort
basso
Conseguenze se non accettata
L'applicazione per essere utilizzata dovrà ogni volta essere configurata e ospitata su un server gestito dall'utente.
Da fare dopo
CR_2

Table 3: CR_3: Deploy dell'app su una piattaforma di hosting online.

1.3.1 Metodologia

Si utilizzerà un servizio di hosting gratuito dove verrà caricata l'applicazione opportunamente configurata.

1.3.2 Risultati attesi

Se la change request fosse approvata, il tool beneficerebbe di diverse migliorie, tra cui:

- **Disponibilità continua del servizio**: Pubblicando l'applicazione e i servizi su una piattaforma di hosting online, si garantisce una disponibilità continua

del servizio agli utenti. La piattaforma di hosting gestirà l'infrastruttura sottostante, garantendo che l'applicazione sia sempre accessibile, anche in caso di picchi di traffico o problemi tecnici.

- **Affidabilità**: La piattaforme di hosting offre un'infrastruttura affidabile e scalabile, garantendo un'elevata disponibilità dell'applicazione e riducendo il rischio di downtime non pianificato.
- **Monitoraggio e gestione**: Le piattaforme di hosting offrono strumenti integrati per il monitoraggio delle prestazioni, la gestione dei log e l'analisi dei dati, consentendo di identificare e risolvere rapidamente eventuali problemi e ottimizzare le prestazioni dell'applicazione.
- **Aggiornamenti e manutenzione**: La piattaforme di hosting online semplificano il processo di aggiornamento e manutenzione dell'applicazione, consentendo di applicare patch di sicurezza, aggiornamenti del software e miglioramenti delle prestazioni in modo rapido e senza interruzioni del servizio.

2 Analisi di TOAST tramite reverse engineering

Prima di contribuire a TOAST, è stato necessario acquisire una conoscenza più approfondita dell'implementazione dello strumento. Per questo motivo, è stato eseguito il processo di reverse engineering del codice sorgente, al fine di analizzare il software e determinare le varie componenti e le relazioni tra di esse. Inizialmente, è stato estratto un diagramma delle classi del sistema, illustrato in fig:1, per rappresentare il sistema a un livello di astrazione maggiore e aumentarne la comprensibilità; è importante notare che il sistema non segue un classico approccio orientato agli oggetti. Successivamente, sono state comprese le responsabilità di ciascun modulo. L'esecuzione di questo processo è stata suddivisa in due attività: l'estrazione delle dipendenze tra i vari moduli (2.1) e l'estrazione dei requisiti funzionali che, a nostro avviso, hanno guidato il progettista nella creazione dello strumento (2.2).

A causa dell'assenza di documentazione, è stato necessario fare affidamento esclusivamente sul codice sorgente. Di conseguenza, non sarà necessario preoccuparsi della tracciabilità tra gli artefatti durante l'esecuzione delle modifiche.

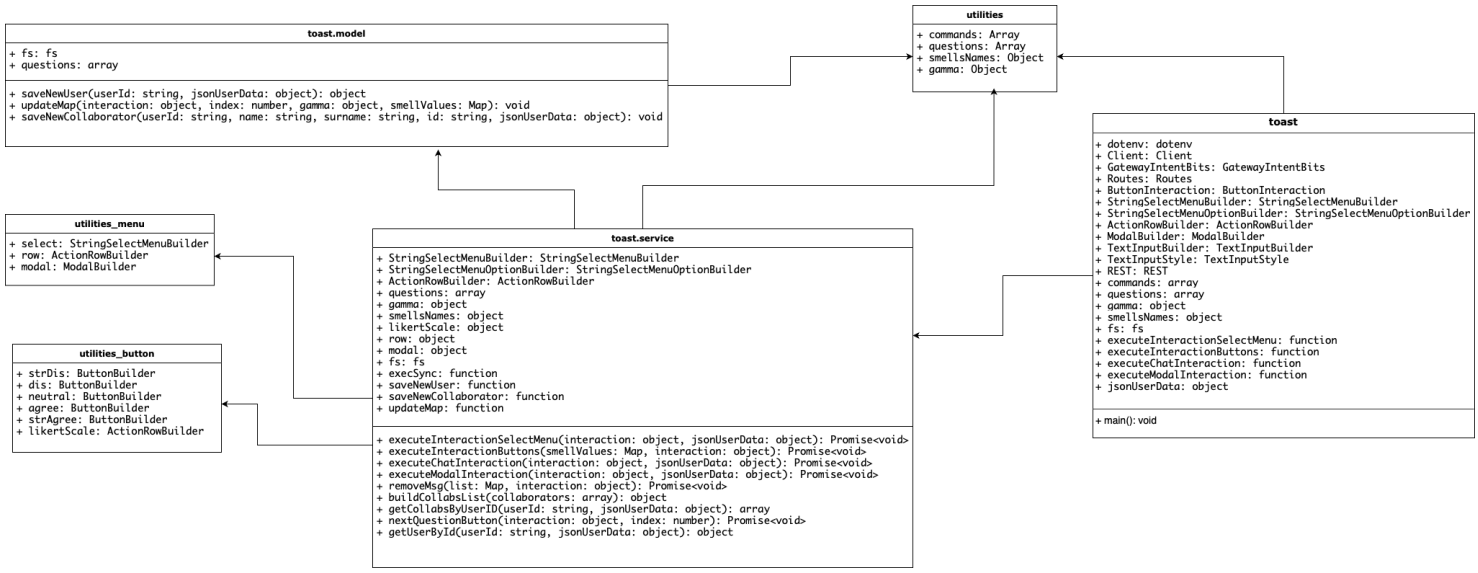

Figure 1: Class diagram di TOAST.

2.1 Estrazione delle dipendenze

Il primo passo intrapreso è stato l'estrazione delle dipendenze tra i moduli del sistema. È stato analizzato il codice sorgente modulo per modulo, al fine di ottenere una conoscenza approfondita dell'implementazione del sistema. È stata creata una matrice che mostra quale modulo utilizza una funzionalità di un altro, indicando nella cella corrispondente il numero di volte in cui ciò accade. Ad esempio, il file **toast-service.js** utilizza **toast.model.js** quattro volte.

La matrice evidenzia che il modulo maggiormente dipendente dagli altri è **toast.service.js**, come chiaramente visibile dalla prima riga. Il modulo **toast.js** contiene la funzione principale che avvia il tool e presenta quattro dipendenze da **toast.service.js**, che viene utilizzato per gestire le interazioni con il tool. Gli altri moduli (**utilities.js**, **utilities.menu.js** e **utilities.button.js**) non utilizzano funzionalità del resto del sistema e si occupano di definire gli elementi dell'interfaccia utente, le cui funzionalità vengono invece utilizzate da **toast.service.js**. Il modulo **toast.model.js** gestisce la persistenza tramite un file JSON, e le sue funzionalità sono anch'esse utilizzate dal modulo **toast.service.js**.

Modulo	Id	1	2	3	4	5	6	7
toast.service	1	0	4	0	5	2	1	0
toast.model	2	0	0	0	1	0	0	2
toast	3	4	0	0	1	0	0	1
utilities	4	0	0	0	0	0	0	0
utilities.menu	5	0	0	0	0	0	0	0
utilities.button	6	0	0	0	0	0	0	0

Table 4: Matrcice delle dipendeze tra i moduli di TOAST

2.2 Estrazione dei requisiti

Attraverso l'analisi del codice sorgente e l'osservazione dell'esecuzione dello strumento, sono stati identificati i requisiti funzionali del tool originale.
Il tool, eseguibile tramite Discord, dopo una breve introduzione e descrizione delle sue funzionalità, consente all'utente (manager) di eseguire due principali operazioni: l'aggiunta di un collaboratore a un team e l'analisi di un collaboratore. Selezionando la prima opzione, verrà aperta una nuova finestra che permetterà l'inserimento di alcuni input testuali, rappresentanti i dati del collaboratore: NOME, COGNOME e ID. Tali dati verranno inviati al server Node per l'elaborazione. Il server preleverà automaticamente i dati rilevanti del manager, quali ID, USERNAME, CHANNEL_ID, USER_ID, e, utilizzando i dati inviati dall'applicazione Discord, creerà e aggiungerà i collaboratori al team corrispondente. Per questa operazione, il tool utilizza un file di persistenza JSON denominato user.json.
La seconda operazione che l'utente può effettuare è l'analisi di uno dei collaboratori del proprio team. Per procedere, l'utente seleziona l'opzione nel menu di selezione, che visualizzerà un questionario contenente una serie di domande con cinque possibili risposte (pulsanti). Al termine del questionario, il tool analizzerà i dati raccolti e mostrerà i valori corrispondenti del collaboratore in relazione a tre community smells: Lone Wolf, Prima Donna e Black Cloud. Pertanto, sono stati individuati tre requisiti funzionali principali:

1. **RF_1 Aggiunta di un collaboratore**: Il sistema deve permettere l'aggiunta di uno o più collaboratori al team gestisto dal manager.

2. **RF_2 Analisi del collaboratore**: Il sistema deve permettere l'analisi del

collaboratore tramite questionario al fine di rilevare eventuali community smell.

3. **Gestione della persistenza**: Il sistema deve permettere la memorizzazione dei dati relativi agli utenti (manager e collaboratori). I dati devono essere persistenti nel sistema.

3 Impact Analysis

Prima di procedere con l'implementazione delle modifiche al progetto, è essenziale condurre uno studio approfondito e un'analisi dettagliata per valutare il potenziale impatto delle stesse sul resto del sistema. Pertanto, è necessario esaminare attentamente come ogni cambiamento possa influenzare l'insieme dei moduli e dei file coinvolti nell'applicazione. Data l'assenza di una documentazione formale, ci si basa esclusivamente sul codice dell'applicazione, sui commenti inseriti al suo interno e sulle tabelle/diagrammi precedentemente generati. Da questa fase di analisi, si genera un insieme iniziale di impatti, denominato **Starting Impact Set**.

3.1 CR_1: Creazione di API per i servizi di TOAST.

Per individuare gli insiemi di moduli impattati, per prima cosa si considerano i requisiti funzionali precedentemente individuati. Visto che le API devono riguardare i servizi per l'aggiunta e l'analisi di un collaboratore i requisiti funzionali impattati sono RF_1 e RF_2. Si individua quindi lo Starting Impact Set formato da i moduli coinvolti direttamente:

- **Toast.js** : è il modulo principale invocato quando l'utente avvia il tool.
- **Toast.model.js** : è il modulo che si occupa di aggiungere i collaboratori al file di persistenza e di effettuare l'analisi di un collaboratore.

Il modulo Toast.service.js non è stato incluso nel SIS in quanto, si occupa unicamente di gestire le interazioni con il tool attraverso l'interfaccia di discord. Le operazioni di aggiunta/analisi vengono effettuate attraverso il modulo: Toast.model.js.
Tenendo conto della matrice delle dipendenze, si analizzano tutti i moduli che possono essere impattati direttamente o indirettamente paretndo dal SIS, ovvero utilities.js, utilities_button.js, utilities_menu.js,toast.service. Essi, possono essere esclusi in quanto fungono da moduli ausiliari per l'estetica e la creazione di elementi grafici per l'applicazione discord e per l'interazione tra l'utente e il bot.

A questo punto il CIS (Candidate Impact Set) risulta uguale al SIS. L'AIS (Actual Impact Set) sarà invece definito dopo l'implementazione.

3.2 CR_2: Sviluppo di un'applicazione web con interfaccia grafica migliorata.

La creazione delle API per i servizi di TOAST consente di disaccoppiare il tool dalla piattaforma Discord, permettendo l'utilizzo dei suoi servizi senza la necessità di modificare direttamente il tool stesso (Aggiunta/analisi collaboratori). Dopo un'attenta analisi del sistema, si è constatato fosse opportuno aggiungere una nuova funzionalità nella web app. Si è ritenuto adeguato aggiungere un diverso tipo di Login per gli utenti sprovvisti di account discord. Dunque l'applicazione supporterebbe un login classico attraverso le Api di autenticazione di discord ed una più "classica" (email, password). A questo punto, si riesegue l'impact analysis con la stessa metodologia utilizzata precedentemente.
Il sistema a questo punto deve gestire due tipi di utenti: (utente web e utente discord). Il sistema dovrà gestire la creazione e l'interazione con il nuovo tipo di utente. Di conseguenza RF_3 sarà impattato.
Si individua quindi lo Starting Impact Set formato da i moduli coinvolti direttamente:

- **Toast.js**
- **Toast.model.js**

Tenendo conto della matrice delle dipendenze, si analizzano tutti i moduli che possono essere impattati direttamente o indirettamente paretndo dal SIS, ovvero utilities.js, utilities_button.js, utilities_menu.js,toast.service. Questi moduli possono essere sclusi in quanto non si occupano della gestione della persistenza e non interagiscono direttamente con il file user.json.

3.3 CR_3: Deploy dell'app su una piattaforma di hosting online.

Durante l'analisi del tool, è stato constatato che l'aggiunta di tale modifica non avrà un impatto diretto sul sistema esistente. Infatti, per implementare questa modifica è sufficiente generare una build del sistema in modo che sia adatta al deploy online, e successivamente caricare i file del sistema in un servizio di hosting. Di conseguenza non sono stati identificati lo Starting Impact Set ed il Candidate Impact Set.

4 Implementazione dei cambiamenti

Dopo aver analizzato i possibili impatti delle modifiche sul sistema, abbiamo effettuato il fork del repository originale su GITHUB per poter fornire i nostri contributi. In questa sezione forniremo un breve riepilogo delle tecnologie utilizzate nel progetto e di come funzionano, sin dalla comprensione dei pattern di codice e del linguaggio di programmazione ha svolto un ruolo enorme nel processo di manutenzione ed evoluzione, quindi descriveremo in dettaglio l'implementazione delle modifiche e il loro risultato.

4.1 Analisi del progetto esistente

Il progetto originale contiene moduli javaScript che sono fortemente legati all'utilizzo dell'interazione dell'utente per stabilire il flusso di esecuzione del tool, effettuare l'analisi degli smells e aggiungere nuovi collaboratori.
Il modulo **Toast.js** in base al tipo di interazione chiama le funzioni di **Toast.service.js**.
Per gestire l'analisi il modulo service usa una funzione **executeInteractionButton** che prende come input un oggetto interaction che rappresenta l'interazione attuale dell'utente e un oggetto Map con i risultati degli smell ottenuti fino all'interazione corrente, la funzione chiama a sua volta **updateMap** (inclusa in toastModel.js), updateMap prende come parametri: l'interazione, un indice globale, gamma(un oggetto che specifica i valori associati alle possibili risposte) e smellsValue (una mappa per memorizzare i risultati), la funzione updateMap viene chiamata ad ogni interazione fino a che tutte le domande sono state sottoposte all'utente, a quel punto l'indice globale viene rimesso a zero. Analizzando updateMap si capisce che questa utilizzi l'interazione per memorizzare l'id del collaboratore, così da associare a lui i risultati, aggiornarli nelle interazioni(risposte) successive e per avere la risposta dell'utente (interaction.**customId**) . La funzione principale che si occupa di analizzare l'utente è proprio updateMap: essa calcola un valore in base a:

- peso della risposta fornita (con $0 \leq \gamma \leq 1$)
- un peso associato alla domanda
- valore calcolato nella precedente interazione

L'aggiunta dei collaboratori viene invece gestita dal moulo toast.js come interazione **isModalInteraction**, in questo caso viene chiamata la funzione **executeModalInteraction** (nel modulo service).
La funzione oltre a gestire l'interfaccia del bot discord, chiama **saveNewCollaborator**(toast.model) che prende come parametri: id

dell'utente,nome, cognome e id del nuovo collaboratore oltre al file json con tutti gli utenti e collaboratori.

4.2 CR_1: Creazione di API per i servizi di TOAST.

Per craere le API sono state aggiunte delle rotte al modulo toast.js, le API possono bypassare le logiche dell'interfaccia sfruttando direttamente le funzioni messe a disposizione da toast.model.js, ragionando in questo modo è stata creata una rotta per eseguire l'analisi di un collaboratore e una per aggiungere un nuovo collaboratore.

4.2.1 Gestione autenticazione

Per identificare l'utente è stata creata una rotta per effettuare il login tramite discord web, ad autenticazione effettuata un token viene generato e aggiunto alla sessione in modo da poter identificare l'utente (o app esterna) che aggiunge nuovi collaboratori.

4.2.2 Sviluppo di /analyzeCollaboratorAPI

Ricezione dei dati dal client: Il client (solitamente una pagina web front-end o un'app esterna) invia una richiesta POST al percorso `/analyzeCollaboratorAPI`. Il corpo di questa richiesta contiene informazioni relative al collaboratore, in particolare le risposte a domande (come `ans1`, `ans2`, ecc.).
Estrazione dei dati di interazione: Il corpo della richiesta (`req.body`) contiene sei risposte:

- `interactionData.ans1`
- `interactionData.ans2`
- `interactionData.ans3`
- `interactionData.ans4`
- `interactionData.ans5`
- `interactionData.ans6`

Queste risposte vengono raccolte in un array chiamato `customIds`, che rappresenta le risposte fornite per l'analisi del collaboratore.

Validazione dell'input: L'API verifica che tutti e sei i campi richiesti (`ans1`, `ans2`, ecc.) siano presenti. Se manca uno qualsiasi di questi campi, risponde con uno stato 400 e un messaggio di errore: `"Campi richiesti mancanti"`.
Simulazione delle interazioni: La funzione crea un set simulato di interazioni. Un oggetto di interazione fittizio viene creato per ciascuna delle sei risposte utilizzando lo `userId` (impostato su "userGenerico"). Questo oggetto include lo `userId` e il `customId` (che rappresenta la risposta) per ogni interazione. Queste interazioni fittizie vengono quindi passate alla funzione `updateMap`, che elabora e aggiorna una mappa (`simMap`) con i dati rilevanti.
Estrazione dei risultati: La mappa `simMap` contiene valori che rappresentano l'analisi del collaboratore. L'analisi del collaboratore è memorizzata in una variabile `values`, che viene recuperata utilizzando lo `userId`. Ogni valore rappresenta un certo "smell" associato al collaboratore, identificato da un acronimo (`smellAcr`) e dal suo valore corrispondente (`smellValue`).
Formattazione dell'output: Gli acronimi degli "smells" vengono sostituiti con i loro nomi completi utilizzando una mappatura predefinita (`smellsNames`). I dati risultanti sono strutturati come un array di oggetti, ciascuno contenente:

- `smellName`: Il nome completo dello smell.
- `smellValue`: Il valore associato allo smell.

Invio della risposta JSON: L'API risponde con un oggetto JSON contenente gli "smells" analizzati:

```
{
  "smells": [
    { "smellName": "Smell 1", "smellValue": "Value 1" },
    { "smellName": "Smell 2", "smellValue": "Value 2" },
    ...
  ]
}
```

Questo output è pronto per essere utilizzato dal front-end o da un altro sistema esterno per visualizzare i risultati dell'analisi.

4.2.3 Sviluppo di /saveCollaboratorAPI

La funzione `/saveCollaboratorAPI` gestisce una richiesta POST che consente di salvare un nuovo collaboratore associato a un utente nel sistema. I dettagli del collaboratore includono:

- **userId**: ID dell'utente che aggiunge il collaboratore.

- **collaboratorId**: ID univoco del collaboratore.
- **name**: Nome del collaboratore.
- **surname**: Cognome del collaboratore.

Inoltre, è richiesto un token di autenticazione per garantire che l'utente che effettua la richiesta sia autorizzato.
Processo di autenticazione:

- **Token di sessione**: Il sistema verifica l'esistenza di un token di sessione generato al momento dell'autenticazione dell'utente.
- **Token dall'app web**: Se il token non è presente nella sessione, può essere passato come parametro nella richiesta POST (ad esempio, da un'applicazione web esterna).
- **Validazione del token**: Viene chiamata la funzione `validateAuthToken(token)` per verificare la validità del token e assicurarsi che l'utente sia autorizzato ad aggiungere collaboratori. Se il token non è valido, viene restituito un errore di autenticazione con codice di stato 401.

Controllo dei parametri richiesti La funzione verifica che i parametri essenziali (`userId`, `collaboratorId`, `name`, `surname` e `token`) siano presenti nella richiesta. Se uno o più di questi campi mancano, la funzione restituirà un errore con codice di stato 400.
Salvataggio del collaboratore: Il file `users.json` viene letto per ottenere i dati degli utenti già presenti nel sistema. La funzione `saveNewCollaborator` viene utilizzata per aggiungere il nuovo collaboratore, associandolo all'utente specificato (`userId`). Questa funzione aggiorna i dati locali e salva il collaboratore nel file JSON.
Risposta JSON: Se il salvataggio del collaboratore è andato a buon fine, la funzione restituirà una risposta JSON con i seguenti campi:

- **success**: Un flag che indica il successo dell'operazione (`true`).
- **message**: Un messaggio di conferma che specifica che il collaboratore è stato aggiunto con successo.
- **collaborator**: Un oggetto che include i dettagli del collaboratore appena aggiunto (`id`, `name` e `surname`).

Errori e gestione delle eccezioni: Se si verifica un errore nel processo (ad esempio, nella lettura del file o nel salvataggio), la funzione restituirà una risposta di errore con codice di stato 500 e un messaggio generico che segnala un problema nel salvataggio del collaboratore.
Esempio di risposta JSON di successo:

```
{
  "success": true,
  "message": "Collaboratore con id: 12345 aggiunto correttamente",
  "collaborator": {
    "id": "12345",
    "name": "Mario",
    "surname": "Rossi"
  }
}
```

4.3 CR_2: Sviluppo di un'applicazione web con interfaccia grafica migliorata.

L'applicazione web è stata sviluppata attraverso il framework express.js. Per visualizzare le varie pagine (home,profile,newCollaborator ecc) , sono state create templates html disposti nella cartella **templates**. Per migliorare l'interazione, l'usabilità e l'aspetto, le pagine sono state arricchite con codice css e javascript (inseriti nella cartella **static** del progetto). Per differenziare le tipologie di login, (leggi 3.1) sono state aggiunte rotte back-end per il login standard (web) e la registrazione.

4.3.1 Sviluppo di /register

La funzione riceve i dati dell'utente tramite il corpo della richiesta (`req.body`). Chiama la funzione `saveNewUserFromWeb` per memorizzare il nuovo utente nel sistema. Attende il risultato di questa funzione asincrona e gestisce il successo o l'errore in base alla risposta:

- In caso di successo: restituisce uno stato 201 e un messaggio di conferma.
- In caso di errore: registra l'errore e invia uno stato 500 con un messaggio di errore.

Sviluppo di saveNewUserFromWeb

La responsabilità di questa funzione è quella di processare e memorizzare i nuovi dati dell'utente (provenienti dall'interfaccia web).

- **Validazione:** Garantisce che i dati necessari dell'utente (ad esempio, email, password) siano presenti.
- **Verifica di Unicità:** Controlla se l'utente esiste già.
- **Generazione id**: Dato che l'utente web non dispone di un Discord id, esso viene gnerato dalla funzione **generateId()**.
- **Memorizzazione dei Dati:** Se i dati sono validi, memorizza i dati dell'utente nel file di persistenza `users.json`).
- **Gestione degli Errori:** Gestisce eventuali errori, come il fallimento della validazione o problemi di lettura/scrittura del file, restituendo risposte appropriate per ciascun caso.

4.3.2 Sviluppo di /analyze

La rotta analyze tramite AXIOS chiama l'API analyzeCollaboratorAPI passando i parametri della richiesta POST (contiene le risposte alle domande per l'analisi), la risposta dell'API(in formato JSON) viene utilizzata per generare una pagina result.html che mostra i risultati dell'analisi effettuata.

4.3.3 Sviluppo di /addCollaborator

La rotta addCollaborator prende i parametri passati nella richiesta POST per aggiungere un nuovo collaboratore, aggiunge l'id dell'utente e un token alla richiesta, tramite axios chiama l'API saveNewCollaboratorAPI, la risposta dell'API viene utilizzata per generare una pagina html personal_area.html da mostrare all'utente (la pagina conterrà un messaggio per confermare l'aggiunta del collaboratore e una card con i dati del collaboratore sarà visibile.

4.3.4 Sviluppo delle rotte restanti

Le altre rotte aggiunte vengono utilizzate per visualizzare le pagine dell'app con i necessari parametri, ad esempio /profile permette di visualizzare la pagina personal_area.html con i dati dell'utente: userId, email e username.

4.4 CR_3: Deploy dell'app su una piattaforma di hosting online.

Per il deploy dell'applicazione, è stata utilizzata la piattaforma di hosting Render: render.com. Dopo aver effettuato l'accesso alla piattaforma, mediante il servizio di autenticazione di GitHub, l'account è stato collegato per consentire la selezione del repository corretto contenente il progetto. Successivamente, sono stati configurati i parametri necessari, come il nome del servizio, l'ambiente di runtime (in questo caso Node.js), i comandi di build per l'installazione delle dipendenze e i comandi di avvio per eseguire il file di avvio definito nel file package.json del progetto. Inoltre, sono state impostate le variabili d'ambiente necessarie al corretto funzionamento dell'applicazione (come il token di Discord, il client_id, ecc.). Infine, avviando il processo di deploy, la piattaforma ha eseguito automaticamente la build del sistema, gestito l'installazione delle dipendenze e generato un URL pubblico per accedere all'applicazione distribuita.

5 Risultati ottenuti

In questa sezione verranno mostrati i risultati ottenuti implementando le modifiche al sistema.

5.1 CR_1: Creazione di API per i servizi di TOAST.

L'implementazione di questa modifica è stata completata con successo e i risultati ottenuti corrispondono alle aspettative. Le chiamate API consentono di utilizzare il tool tramite richieste POST, permettendo l'accesso ai servizi di analisi e gestione dei collaboratori. Per quanto riguarda l'impatto delle modifiche non sono stati rilevati nuovi moduli impattati, dunque l'Actual Impact Set (AIS) corrisponde al Candidate Impact Set (CIS).

$$\text{Recall} = \frac{|CIS \cap AIS|}{|AIS|} = \frac{2}{2} = 100\%$$

$$\text{Precision} = \frac{|CIS \cap AIS|}{|CIS|} = \frac{2}{2} = 100\%$$

5.2 CR_2: Sviluppo di un'applicazione web con interfaccia grafica migliorata.

L'implementazione di questa modifica è stata completata con successo e i risultati ottenuti corrispondono alle aspettative. L'applicazione web risulta funzionante e gli obiettivi di usabilità e miglioramento grafico sono stati rispettati. Adesso gli utenti possono interagire con il tool tramite un'interfaccia pensata appositamente. Per quanto riguarda l'impatto delle modifiche non sono stati rilevati nuovi moduli impattati, dunque l'Actual Impact Set (AIS) corrisponde al Candidate Impact Set (CIS).

$$\text{Recall} = \frac{|CIS \cap AIS|}{|AIS|} = \frac{2}{2} = 100\%$$

$$\text{Precision} = \frac{|CIS \cap AIS|}{|CIS|} = \frac{2}{2} = 100\%$$

5.3 CR_3: Deploy dell'app su una piattaforma di hosting online.

Il processo di deploy è stato effettuato con successo, l'applicazione è disponibile al seguente URL: https://toast-tool.onrender.com/