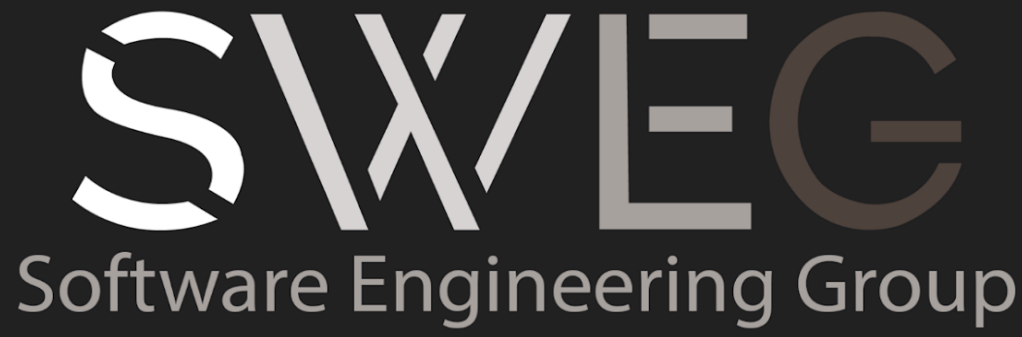

swegsoftware@gmail.com

Specifica Tecnica

Informazioni sul documento

Redattori:	Davide M.	Gabriel R.	Marco B.
Verificatori:	Andrea M.	Davide S.	Davide M.
Approvazione:	Davide M.		
Destinatari:	T. Vardanega	R. Cardin	Zero12
Versione:	1.0.0		
Uso:	Esterno		

Registro dei Cambiamenti - Changelog

Versione	*Data*	*Autore*	*Verificatore*	*Dettaglio*
1.0.0	2023-05-17	Davide Milan	Gabriel Rovesti	Approvazione
0.9.6	2023-05-10	Gabriel Rovesti	Davide Milan	Correzioni di massima e modifica vari UML esistenti
0.9.5	2023-05-09	Marco Bernardi	Davide Milan	Inserimento sezione chiamate API e relative descrizioni
0.9.0	2023-04-28	Gabriel Rovesti	Davide Milan	Aggiunta ultimi UML, descrizione schema base di dati
0.8.0	2023-04-27	Davide Sgrazzutti	Davide Milan	Aggiunta UML diagrammi delle classi e descrizioni testuali e strutturazione sezioni
0.7.0	2023-04-26	Gabriel Rovesti	Marco Bernardi	Aggiunta UML diagrammi delle classi e descrizioni testuali e strutturazione sezioni
0.6.0	2023-04-24	Gabriel Rovesti	Marco Bernardi	Scrittura tabella requisiti soddisfatti, impostazione sezioni e creazione sezioni architetturali frontend e backend
0.5.0	2023-04-24	Gabriel Rovesti	Marco Bernardi	Ampliamento sezioni front-end e back-end e scrittura indici relativi, introduzioni e ampliamento API
0.4.0	2023-04-19	Marco Bernardi	Davide Sgrazzutti	Scrittura sezione 4
0.3.0	2023-04-15	Davide Milan	Andrea Meneghello	Creazione indici sezioni 3,4 e 5 e prima scrittura sezione 3
0.2.0	2023-04-10	Gabriel Rovesti	Andrea Meneghello	Scrittura sezione 2 e sottosezioni
0.1.0	2023-03-20	Davide Milan	Davide Sgrazzutti	Inizializzazione documento e creazione indici e struttura

Sommario

Elenco delle immagini

Elenco delle tabelle

Elenco dei grafici

1 Introduzione

1.1 Scopo del documento

Lo scopo di questo documento è descrivere e motivare tutte le scelte architetturali che sono state fatte nella fase di progettazione e codifica del prodotto.
Vengono quindi riportati i diagrammi dei React Components[G] e dei package per descrivere le scelte di design pattern utilizzate per realizzare l'architettura finale del prodotto.

È poi presente una sezione dedicata ai requisiti che il gruppo è riuscito a soddisfare, così da fornire un'ampia visione sullo stato di avanzamento del lavoro.

1.2 Scopo del prodotto

L'obiettivo del progetto è realizzare una piattaforma per la gestione delle traduzioni tramite un'infrastruttura multi-tenant[G], permettendo di gestire testi in lingue diverse e a diversi utenti. Esso si rivolge a siti che offrono la possibilità di essere visualizzati in lingue differenti e gestire bilateralmente le traduzioni dei testi che li compongono, secondo un meccanismo di accesso, collaborazione, approvazione e rifiuto.

La piattaforma consente agli utenti di accedere al proprio Tenant e interagire a diverso livello con le traduzioni presenti. L'utilizzo delle tecnologie AWS[G] permette di riconoscere il tipo di utente tramite uno specifico token[G] a lui assegnato, permettendogli una specifica gestione della piattaforma. In particolare, possiamo distinguere:

- gli utenti finali (definiti come User[G]) potranno visualizzare un insieme di testi che dovranno tradurre;
- gli utenti amministratori (definiti come Admin[G]), che gestiscono un singolo Tenant, potranno impostare una serie di lingue secondarie, visualizzare e modificare testi originali, approvare e rifiutare le traduzioni presenti. Essi possono inoltre creare, modificare e cancellare categorie di traduzioni presenti e inviare testi tradotti assegnati;
- gli utenti definiti come SuperAdmin[G], con permessi di gestione di tutti i Tenant, possono gestire ciascuno di questi e gli utenti ad essi associati.

Ogni traduzione è raggruppata in una categoria, e le traduzioni sono suddivise tra lingua di default impostata e una serie di lingue secondarie. Inoltre, l'applicativo consente di visualizzare le traduzioni eseguite o non eseguite e di visualizzare i testi tradotti in una lingua tramite API.

Per fornire la massima compatibilità, essa sarà fruibile tramite browser[G], in grado di supportare correttamente le tecnologie di base HTML[G], CSS[G], JavaScript[G] ed altre successivamente definite.

1.3 Glossario

Al fine di evitare incomprensioni relative alla terminologia usata all'interno del documento, viene fornito un Glossario nel file apposito (Glossario v.2.0.0), tale da non avere terminologie ambigue nell'attività progettuale individuata e dandone una definizione precisa. Ogni termine avrà nel documento una lettera G come apice, per meglio evidenziare la loro appartenenza al documento indicato.

1.4 Maturità e miglioramenti

Il presente documento è redatto con un approccio incrementale, al fine di poter implementare facilmente cambiamenti nel corso del tempo a seconda di esigenze concordate bilateralmente tra membri del gruppo e proponente. Pertanto, non può essere considerato definitivo e completo in questa versione.

1.5 Riferimenti

1.5.1 Riferimenti normativi

- Capitolato[G] C4-Piattaforma di localizzazione testi:
  - https://www.math.unipd.it/~tullio/IS-1/2022/Progetto/C4.pdf
- Norme di Progetto v.2.0.0
- Regolamento progetto didattico:
  - https://www.math.unipd.it/~tullio/IS-1/2022/Dispense/PD02.pdf

1.5.2 Riferimenti informativi

- Analisi dei Requisiti v.2.0.0
- Capitolato C4-Piattaforma di localizzazione testi:
  - https://www.math.unipd.it/~tullio/IS-1/2022/Progetto/C4.pdf
- Slide T07 del corso di Ingegneria del Software - Analisi dei requisiti: https://www.math.unipd.it/~tullio/IS-1/2022/Dispense/T06.pdf
- Slide del corso di Ingegneria del Software - Progettazione e programmazione: Diagrammi delle classi:
  - https://www.math.unipd.it/~rcardin/swea/2022/Diagrammi%20Use%20Case.pdf
- Slide del corso di Ingegneria del Software - Solid Programming:
  - https://www.math.unipd.it/~rcardin/swea/2021/SOLID%20Principles%20of%20Object-Oriented%20Design_4x4.pdf

1.5.3 Riferimenti tecnici

- Documentazione di React:
  - https://react.dev/
- NPM:
  - https://docs.npmjs.com/
- Documentazione di TypeScript:
  - https://www.typescriptlang.org/docs/
- Documentazione di AWS:
  - https://docs.aws.amazon.com/

2 Tecnologie

In questa sezione viene fornita una panoramica generale delle tecnologie utilizzate per la realizzazione del prodotto in questione. Vengono infatti descritte le procedure, gli strumenti e le librerie necessari per lo sviluppo, il test e la distribuzione del prodotto. In particolare, verranno trattate le tecnologie impiegate per la realizzazione del front-end e del back-end, per la gestione del database e per l'integrazione con i servizi previsti.

2.1 Tecnologie per la codifica

Tecnologia	***Descrizione***	***Versione***
	Linguaggi	
HTML	Linguaggio di annotazione (markup) utilizzato per impostare la struttura delle singole pagine e definire gli elementi dell'interfaccia	5
CSS	Linguaggio utilizzato per la formattazione e la gestione dello stile degli elementi HTML	3
JavaScript	Linguaggio utilizzato per la gestione di eventi invocati dall'utente,	ECMAScript 2021
TypeScript[G]	Superset di JavaScript per utilizzare tipizzazione	5.0.x
	Librerie e framework[G]	
ReactJS[G]	Libreria grafica per facilitare lo sviluppo front-end gestendo modularmente le componenti grafiche, permettendo performance buone grazie all'efficacia della sua renderizzazione	18.0.x
Material UI[G]	Framework di componenti React preconfezionati per la creazione di interfacce utente gradevoli, funzionali e personalizzabili	4.1.x
	Strumenti e servizi	
Node.js[G]	Ambiente di runtime open-source per l'esecuzione di codice JavaScript lato server tramite appositi script	19.0.x
NPM[G]	Gestore di pacchetti per il linguaggio JavaScript e l'ambiente di esecuzione Node.js	9.6.x
AWS Cognito[G]	Servizio di gestione delle identità ed autenticazione basato sui ruoli	2023-16-02
AWS DynamoDB[G]	Servizio di database non relazionale gestito in modo scalabile	2019-11-21
AWS Lambda[G]	Servizio di elaborazione serverless[G] di codice senza dover configurare/scalare l'infrastruttura server, eseguendo il codice in modo scalabile	2023-03-16
AWS API Gateway[G]	Servizio di gestione delle API[G] che permette di creare, pubblicare e proteggere e le stesse in modo sicuro, concentrandosi sulla logica di business	2023-04-06

Git[G]	Sistema di controllo versione distribuito utilizzato per la gestione del codice sorgente dal parte del gruppo di progetto	2.4.x

Tabella 1 - Tabella di descrizione delle tecnologie

2.2 Tecnologie per l'analisi del codice

Tecnologia	*Descrizione*	*Versione*
	Analisi statica	
ESLint	Strumento JavaScript che aiuta a individuare gli errori di codice e le pratiche non ottimali	8.38.x
Prettier	Strumento di formattazione del codice che aiuta a mantenere uno stile di codifica coerente e leggibile	3.0.x
Jest	Framework di test basato su JavaScript con funzionalità di creazione di mock e il testing del codice in modo asincrono.	29.0.x
	Analisi dinamica	
React Testing Library	Libreria di test integrata nativamente che consente di testare il comportamento dei componenti[G] React da una prospettiva degli utenti finali.	14.0.x
GitHub Actions	Servizio di CI/CD[G] per automatizzare il processo di build[G], test e deploy[G] del progetto software	/

Tabella 2 - Tabella di descrizione degli strumenti di test

3 Architettura

Durante la fase di progettazione, si è deciso di adottare un'architettura a microservizi, permettendo la comunicazione tra gli stessi attraverso le tecnologie AWS[G], in particolare con API Gateway[G]. Secondo le nostre esigenze, non era infatti possibile adottare un pattern architetturale univoco per tutta l'infrastruttura sottostante. Per questo motivo, il sistema è stato convenzionalmente diviso a livello logico e strutturale tra:

- parte front-end, ossia la parte *client* del sistema, eseguibile in locale su qualsiasi browser, sviluppata con l'ausilio di ReactJS e TypeScript;
- parte back-end, sfruttando i diversi servizi forniti da AWS, con l'interazione attraverso NodeJS lato client e successiva comunicazione con la libreria realizzata dal gruppo *SWEG* e attraverso le chiamate definite sulla base delle specifiche esigenze.

La comunicazione tra le singole parti viene realizzata attraverso la tecnologia API Gateway, permettendo al front-end di visualizzare i propri dati e separarli logicamente nella loro gestione, comunicando lato back-end con la libreria scritta in formato *swagger*[G]. Le successive sezioni hanno lo scopo di dettagliare precisamente le singole parti, al fine di descrivere pattern architetturali e scelte di design che il gruppo ha deciso di utilizzare in fase di progettazione.

3.1 Architettura Front-end

3.1.1 Introduzione

L'architettura del prodotto *Translatify* non segue uno specifico pattern architetturale, poiché l'uso di uno specifico non soddisfa a pieno le esigenze di modularità e scalabilità previste dall'applicazione. Si è invece scelto di utilizzare un insieme di design pattern tipici della libreria ReactJS, selezionati ed adattati in base alle specifiche esigenze del progetto, cercando per quanto possibile di garantire la separazione della logica di business tra le varie componenti, semplificando quindi la gestione degli stati dell'applicazione. Ciascuno di questi viene dettagliatamente descritto in sezioni successive.

Un aspetto chiave del prodotto è la modularità raggiunta sulla base dell'implementazione di un contesto di autenticazione che consente di riconoscere globalmente l'utente in base al suo ruolo e quindi recuperare i dati necessari alla visualizzazione nelle diverse pagina attraverso chiamate opportune alle API RESTful realizzate, mostrando solamente le opzioni pertinenti per ogni utente autenticato.

Ogni pagina dell'applicazione utilizza le chiamate alle API per recuperare i dati necessari alla sua visualizzazione e utilizza gli *hooks* forniti da React, come *useState* e *useEffect*, per gestire lo stato dell'applicazione e aggiornare dinamicamente la UI. Grazie a questo approccio, l'applicazione è in grado di fornire un'esperienza utente fluida e reattiva, in grado di adattarsi alle esigenze dell'utente in tempo reale. In particolare, le singole chiamate alle API consentono di recuperare i dati in modo efficiente, sicuro e scalabile, separando la logica di business dall'interfaccia utente. Con l'utilizzo degli hook, invece, è possibile gestire lo stato dell'applicazione in modo dichiarativo e modulare, semplificando la leggibilità del codice e relativa gestione dei dati.

3.1.2 Diagrammi delle Classi

La seguente sezione si occupa di descrivere le singole pagine per i vari utenti utilizzando la sintassi UML[G] e un'opportuna descrizione testuale di completamento, al fine di rendere non ambigua e facilmente interpretabile ogni scelta di progettazione realizzata dal gruppo *SWEG* durante il periodo di sviluppo.

In particolare, si specifica che le successive sezioni saranno così articolate a livello logico/testuale:

- *Viste Generali*, presenti per tutti i tipi di utenti che adottano il sistema di back office multi-tenant[G] realizzato. Questa sezione comprende le pagine apposite di Login e di Recupero Password;
- *Pagine User*, che dettaglia le relative sezioni per lo User[G], comprensive di pagina dei soli testi da tradurre e tutti i testi disponibili nel Tenant a cui è associato
- *Pagine Admin*, che dettaglia le relative sezioni per l'utente Admin[G], comprensive delle stesse pagine viste dall'utente più una serie di pagine relative alla gestione del proprio Tenant di riferimento, relativamente alle categorie di traduzione presenti, testi da approvare/rifiutare e la creazione di un utente associato al proprio Tenant;
- *Pagine SuperAdmin*, che dettaglia le relative sezioni per l'utente SuperAdmin[G], gestore di più Tenant nella sua pagina principale e con la possibilità di crearne di nuovi in ogni momento.

Per evitare ripetizioni nel contenuto, il layout grafico di massima utilizza per le pagine le componenti di MaterialUI *Grid* o *Table* per il layout, oltre all'opportuna impostazione grafica definita meglio sotto. In generale, le pagine utilizzano componenti quali *Typography, Card*, etc. Questi di massima incorporano le componenti grafiche presenti, quali bottoni e altre componenti personalizzate create ad hoc per specifiche funzionalità.

A livello logico e di codice TypeScript, invece, sono stati introdotti dei *tipi* specifici per la gestione dei singoli dati presenti, al fine di fornire maggiore sicurezza del codice scritto e un'esperienza di scrittura del codice più robusta. I tipi vengono utilizzati per definire il tipo di dati di ogni variabile, costante, parametro di funzione e proprietà di oggetto nel codice. Ciò significa che ogni volta che si utilizza una variabile o una funzione, TypeScript controlla il tipo dei dati per garantire che siano corretti e coerenti.

Questo permette di prevenire errori comuni, migliorando la qualità e la manutenibilità del software scritto, rigorosamente definito e non modificabile senza conversione esplicita.

Possiamo quindi dettagliare i seguenti tipi:

- **Tenant**, corrispondente ai dati del Tenant di riferimento, con identificativo, nome, lista degli identificativi degli utenti amministratori, lista degli identificativi degli utenti normali, categorie di traduzione, data di creazione, lingua e lingua predefinita;
- **Text**, corrispondente ai dati di una traduzione o di un testo originale, comprensivo di identificativo del Tenant di riferimento, categoria di traduzione di cui fa parte, titolo, contenuto del testo, stato della traduzione, eventuale feedback,eventuale commento ed eventuale link di riferimento esterno;
- **Category,** che rappresenta una categoria di testo associata a un Tenant. È composta da un identificativo univoco e da un nome;

- **TextState**, costituita da un'enumerazione che categorizza i testi sulla base dei loro stati di traduzioni, rispetto ai nuovi testi da tradurre, testi rifiutati, testi da approvare, testi approvati e testo originale;
- **User**, che rappresenta i dati dell'utente registrato alla piattaforma, comprensiva di username, password, indirizzo email, gruppo di appartenenza per AWS Cognito che permette di identificare il ruolo, nome e cognome.

Il seguente diagramma UML contestualizza quanto appena enunciato:

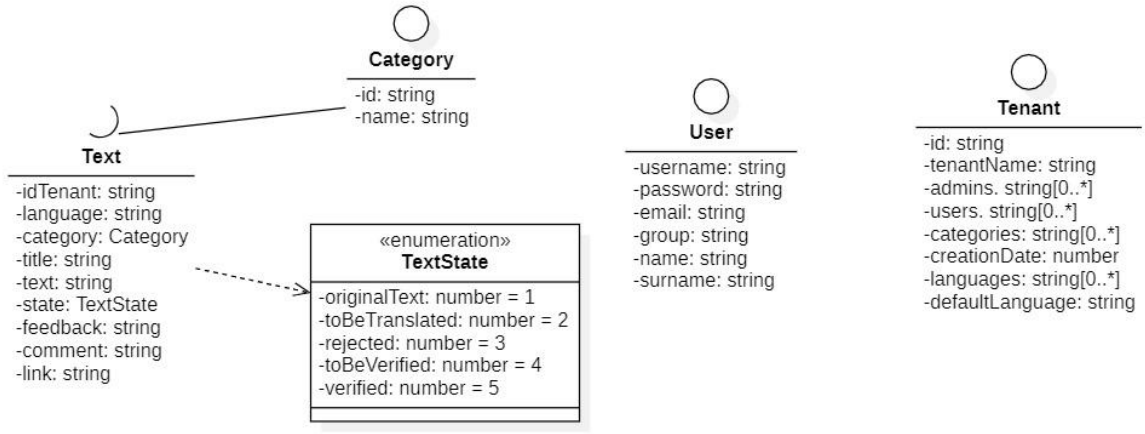

Immagine 1: Tipi TypeScript di riferimento

3.1.2.1 Viste generali

3.1.2.1.1 LoginView

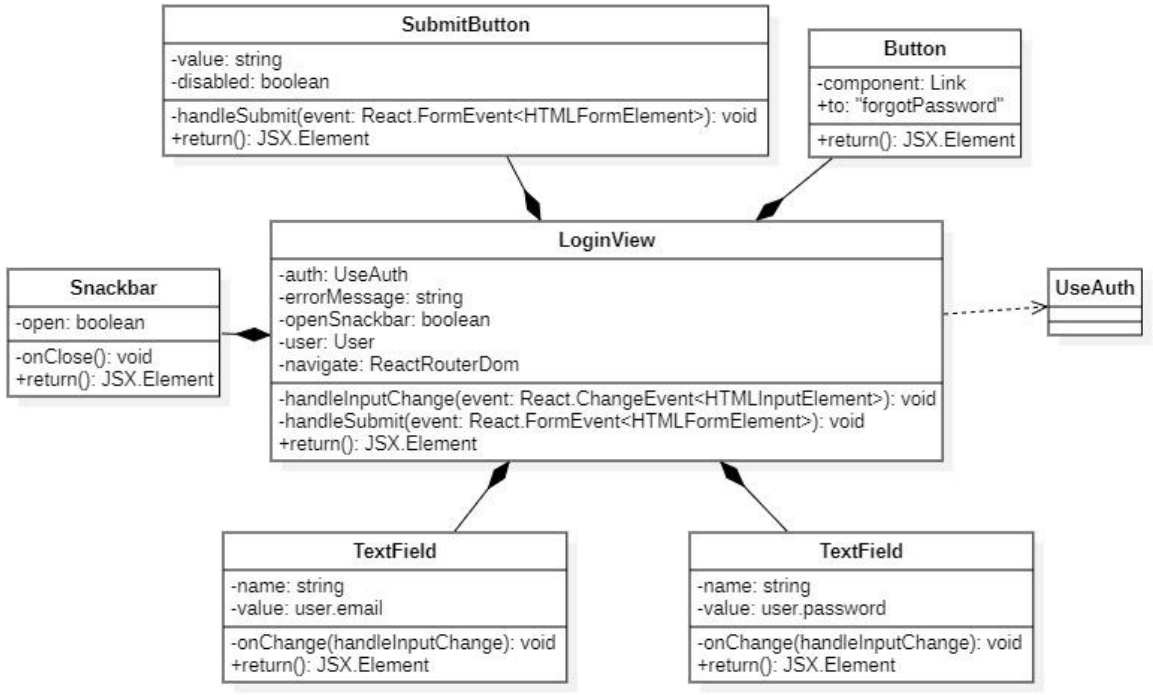

Immagine 2: Diagramma delle classi per la pagina di Login

La pagina *LoginView* utilizza nello specifico due tipologie componenti fornite da Material UI, in particolare *TextField* e *Button*, differenziati ciascuno dalla propria funzionalità. Nello specifico, i componenti di input immettono gli omonimi dati, mentre gli altri due bottoni permettono l'inserimento nel form di riferimento oppure l'opportuno invio alla pagina di recupero password preposta e specificata in questa stessa sezione.

Inoltre, viene utilizzato il contesto di autenticazione *UseAuth*, con la seguente specifica UML:

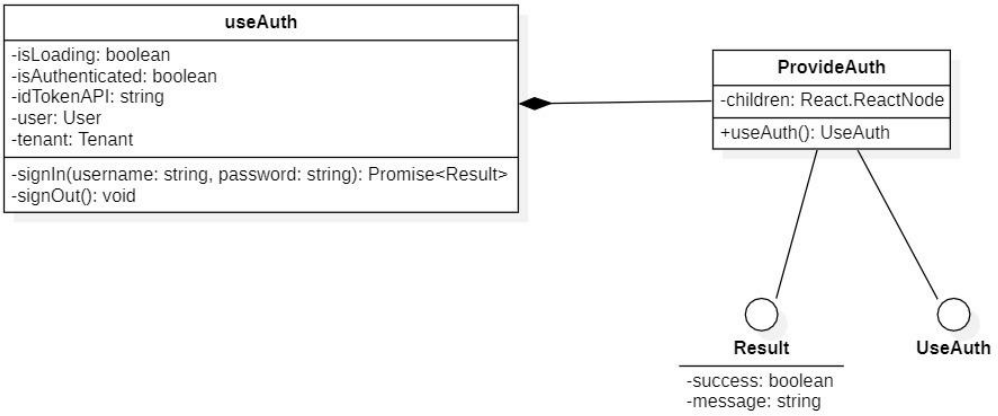

Immagine 3: Diagramma delle classi per il contesto di autenticazione

Il componente permette di definire chiaramente un provider di contesto per l'autenticazione che, definita la configurazione interna di AWS Amplify, permette l'autenticazione ed il recupero delle relative informazioni di contesto. Il parametro di autenticazione permette di associare l'utente, in base al suo tipo, ad un Tenant di riferimento se presente, determinando il successo delle operazioni di login e di logout.

3.1.2.1.2 ForgotPasswordView

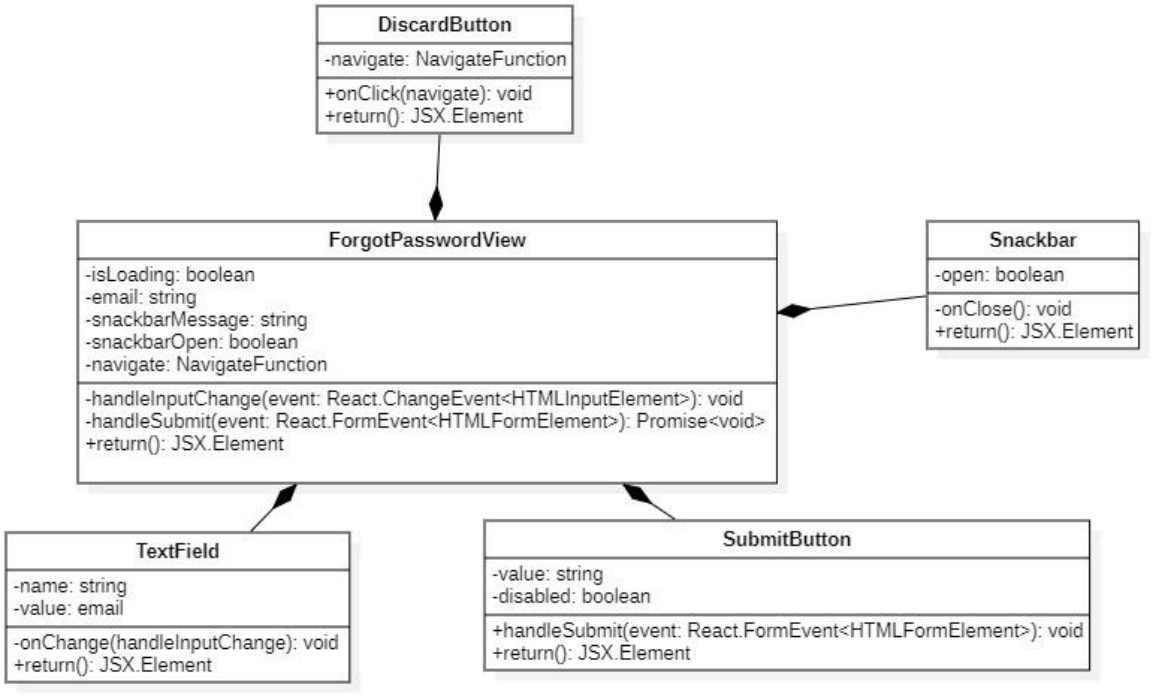

Immagine 4: Diagramma delle classi per la pagina di recupero password

La pagina, comprensiva di visualizzazione di messaggi di errore e di conferma tramite componente *Snackbar* fornito da Material UI, prende i dati al caricamento del componente *TextField* della libreria Material e invia i dati con la relativa funzione del bottone preposto utilizzando le funzioni interne di AWS presenti. Per convenzione, ogni volta che viene eseguito l'invio di dati da un form oppure l'annullamento dell'operazione attuale, sono stati definiti due specifici componenti, denominati *SubmitButton* e *DiscardButton*, il cui utilizzo sarà presente in altre pagine e altri UML qui presenti.

I messaggi di conferma e di errore della pagina vengono gestiti utilizzando i componenti *Snackbar* presenti nella libreria Material UI.

Una volta inserito l'indirizzo email, è presente un'altra pagina che, attraverso una chiamata API, invia il codice di recupero alla mail specificata nel form.

3.1.2.1.3 ResetPasswordView

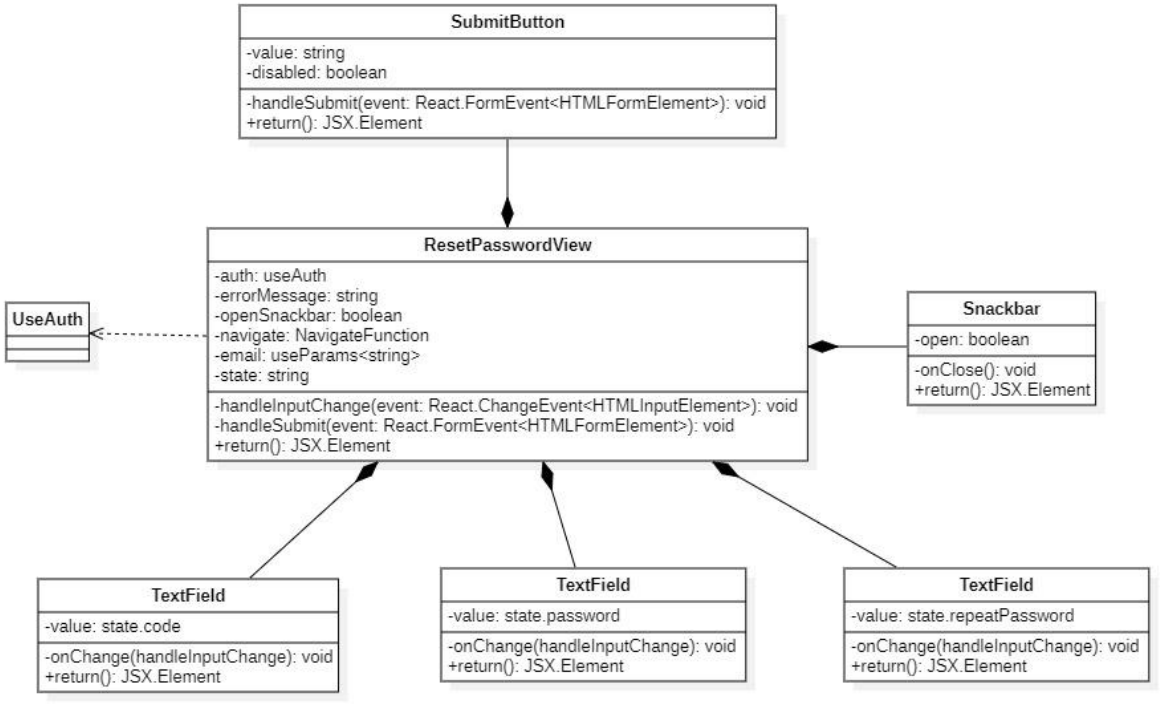

Immagine 5: Diagramma delle classi per la pagina di reset password

La pagina, comprensiva di visualizzazione di messaggi di errore e di conferma tramite componente *Snackbar* fornito da Material UI, prende i dati relativi allo stato di autenticazione tramite i componenti *TextField*, comprensivi di codice recuperato tramite email, password e ripetizione della password, realizzando poi una chiamata API per il reset effettivo nella base di dati presente.

Vengono visualizzati messaggi di errore e di feedback all'utente tramite Snackbar e si utilizzano gli *hooks* per le impostazioni delle relative variabili private presenti.

3.1.2.2 Pagine User

3.1.2.2.1 UserView

Una volta effettuata l'autenticazione, generalmente per ogni pagina sono presenti due componenti importanti:

- *PrivateRoute*, che permette di definire per ogni pagina l'utente o l'insieme di utenti che hanno diritto di accesso alla stessa;
- *LayoutWrapper*, che permette ad un gruppo di utenti riconosciuti in base ai propri permessi usando AWS Cognito. A sua volta, tale componente è composto da:
    - *MainContent*, che permette di impostare uno stile unico usando la componente *Box* fornita da Material UI per le pagine e gli elementi grafici presenti;
    - *UserMenu*, che fornisce la barra di navigazione presente nelle singole pagine e in base al tipo di utente, permette di capire il tipo di utente e le opzioni da visualizzare, compreso il bottone di login.

Il menù utente permette quindi di visualizzare la pagina dei soli testi da tradurre e la pagina di tutti i testi presenti nel Tenant. Questo vale per tutte le pagine previste in questa sezione.

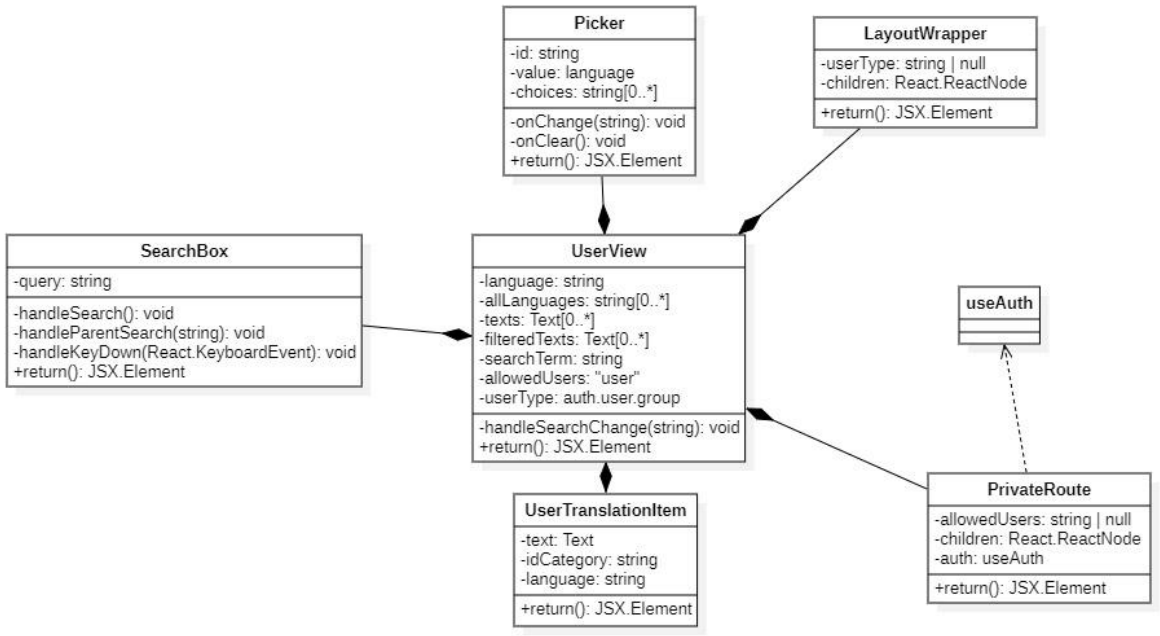

Immagine 6: Diagramma delle classi per la dashboard utente

La pagina rappresenta la dashboard dello User quando autenticato, in particolare visualizzando l'insieme dei soli testi da tradurre, permettendogli di selezionare tra tutti i testi presenti per il Tenant (visti con il componente *TextTranslationItem*) tramite la lingua con il componente *Picker*, una ricerca testuale e filtro sulla base del nome/lingua presente con il componente *TextSearch*. Ogni variabile privata presente nella pagina ha un corrispondente *hook* di impostazione React, tale da poter impostare chiaramente lo stato e quanto presente.

Come per le altre pagine, i messaggi di conferma e di errore della pagina vengono gestiti utilizzando i componenti *Snackbar* presenti nella libreria Material UI.

3.1.2.2.2 TenantTextsView

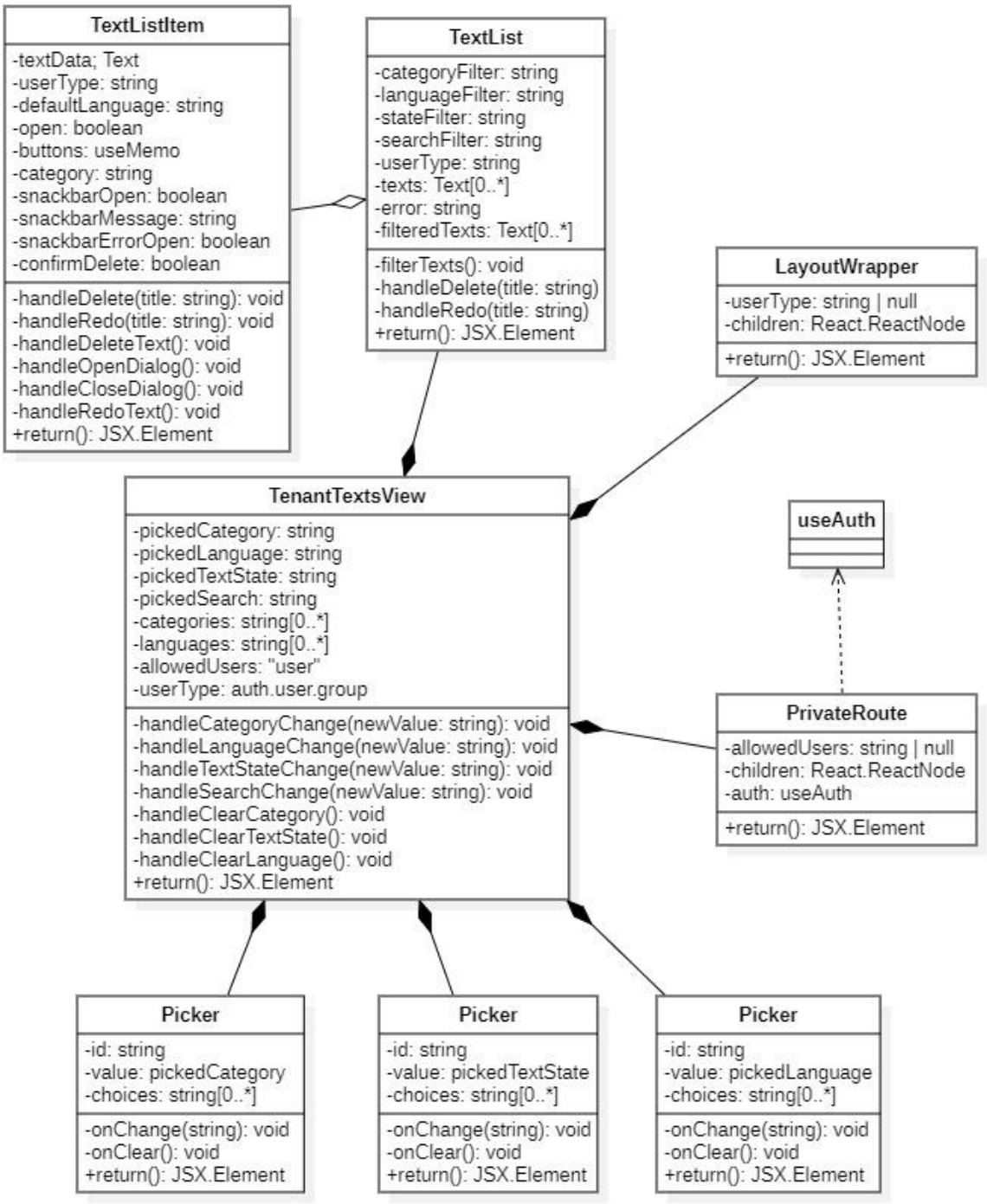

Immagine 7: Diagramma delle classi per la pagina di tutti i testi del Tenant

La pagina presenta tutti i testi previsti per il singolo Tenant, con la possibilità di filtrare i testi, oltre ai soli da tradurre, tra tutti quelli presenti. Di fatto, le opzioni di filtraggio sono previste per lingua, stato e categoria, come previsto dagli opportuni componenti *Picker* e relativi stati di gestione. I testi presenti sono visualizzati come lista dal componente *TextList*, che comprende al suo interno vari *TextListItem* sfruttando il pattern *Compound Components* di React, citato meglio sotto in dettaglio.

Esso prevede il riconoscimento sulla base del tipo di utente, tale da consentire la modifica o la creazione di una nuova traduzione o testo originale quando possibile. Tutte le variabili private prevedono un apposito hook di gestione.

Ogni bottone permette la modifica o la creazione del testo originale o della traduzione, operazioni gestite in due pagine apposite come descritto sotto in maggiore dettaglio.

Similmente a prima, la pagina viene permessa alla navigazione per lo User, quando riconosciuto opportunamente dalle chiamate API e da AWS Cognito con i medesimi componenti *LayoutWrapper* e *PrivateRoute*, incorporando le opzioni di menù previste per il tipo di utente.

L'interazione delle API è prevista con una chiamata di tipo GET attraverso l'utilizzo di uno *useEffect* fornito da React. Come per le altre pagine, i messaggi di conferma e di errore della pagina vengono gestiti utilizzando i componenti *Snackbar* presenti nella libreria Material UI.

3.1.2.2.3 CreateTranslationView

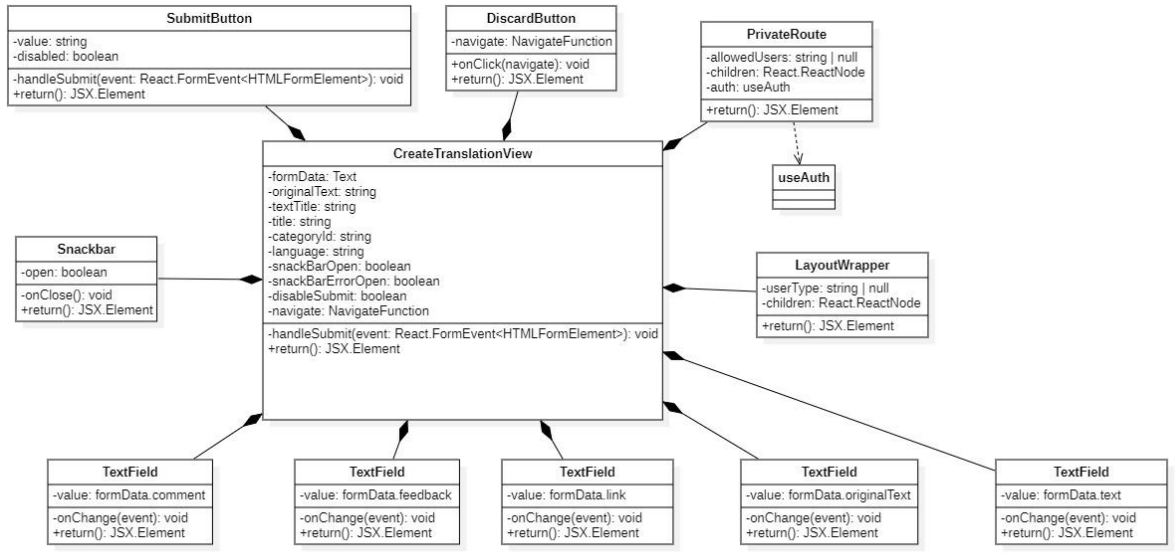

Immagine 8: Diagramma delle classi per la pagina di creazione della singola traduzione

La pagina prevede la creazione di una traduzione, prevedendo il riconoscimento del tipo di utente e una visualizzazione dei campi di un testo originale presenti nei campi *TextField* di Material UI, al fine di permettere la visualizzazione e l'impostazione dei campi presenti, permettendo l'operazione di invio della traduzione in modo semplice e dipendente dall'evento di invio. Tutti i campi della traduzione sono previsti dall'interfaccia *FormState*, che non permette l'invio dei dati finché non viene fornita una traduzione. L'opportuno invio della traduzione o l'annullamento dell'operazione sono opzioni previste tramite i due bottoni *SubmitButton* e *DiscardButton*. Tali operazioni avvengono sempre per l'utente User correttamente autenticato.

L'interazione delle API è prevista con una chiamata di tipo POST attraverso l'utilizzo di uno *useEffect* fornito da React. La libreria *useParams* di React viene utilizzata per indicizzare un testo presentato come elemento grafico e raccogliere i dati di testo e categoria recuperando i dati tramite il link.

Come per le altre pagine, i messaggi di conferma e di errore della pagina vengono gestiti utilizzando i componenti *Snackbar* presenti nella libreria Material UI.

3.1.2.2.4 CreateEditTextView

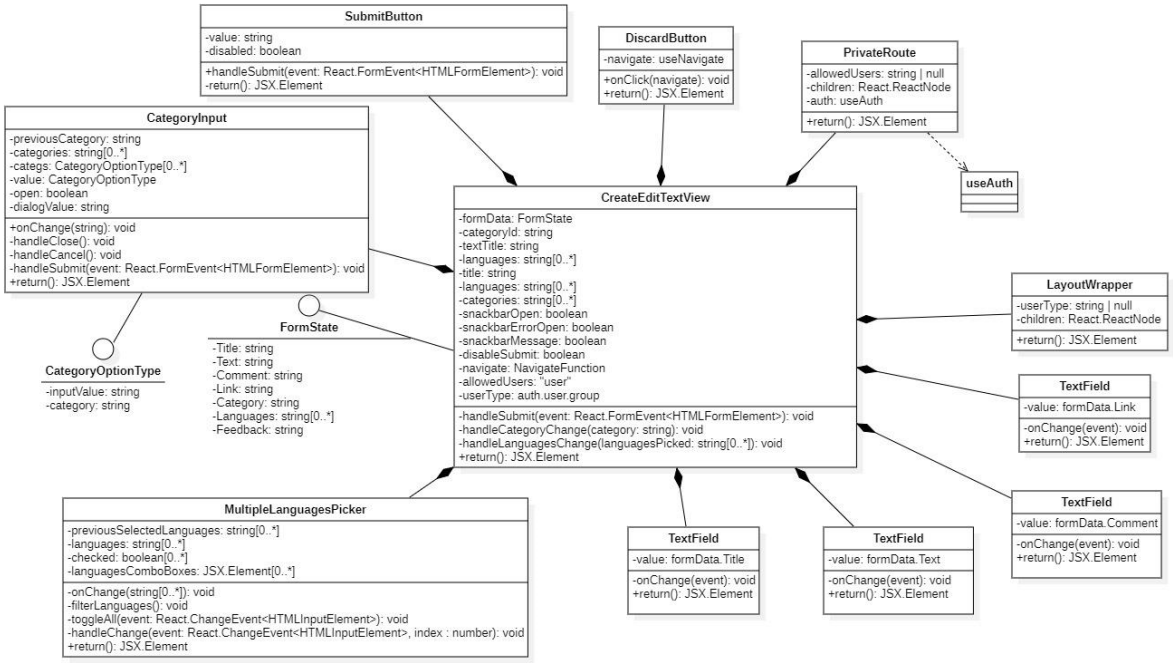

Immagine 9: Diagramma delle classi per la pagina di creazione e modifica del testo originale

La pagina permette di creare o modificare un testo originale, accessibile tramite la pagina di tutti i testi del Tenant. Questa contiene una serie di campi di input per il testo originale, associando anche commenti e link di riferimento associati. Include inoltre un selettore per la categoria tramite il componente *CategoryInput* e l'apposito selettore delle lingue secondarie associate al Tenant, con il componente *MultipleLanguagesPicker*. Tutte le variabili private prevedono un apposito hook di gestione.

Come per le altre pagine, vengono forniti gli appositi pulsanti di invio dei dati del testo originale, eseguendo un'apposita chiamata con una API al server, una volta controllata la validità dei dati inseriti e il fatto dell'accesso protetto, regolato dai corretti permessi per il Tenant di riferimento.

3.1.2.3 Pagine Admin

La componente *PrivateRoute* consente all'utente Admin di poter vedere le stesse pagine dello User ma con i permessi di utente amministratore, come tale creazione, modifica, approvazione e rifiuto dei testi originali. In tal modo, tutte le precedenti pagine definite in precedenza sono accessibili anche all'utente amministratore con permessi ulteriori a seconda del contesto.

3.1.2.3.1 AdminView

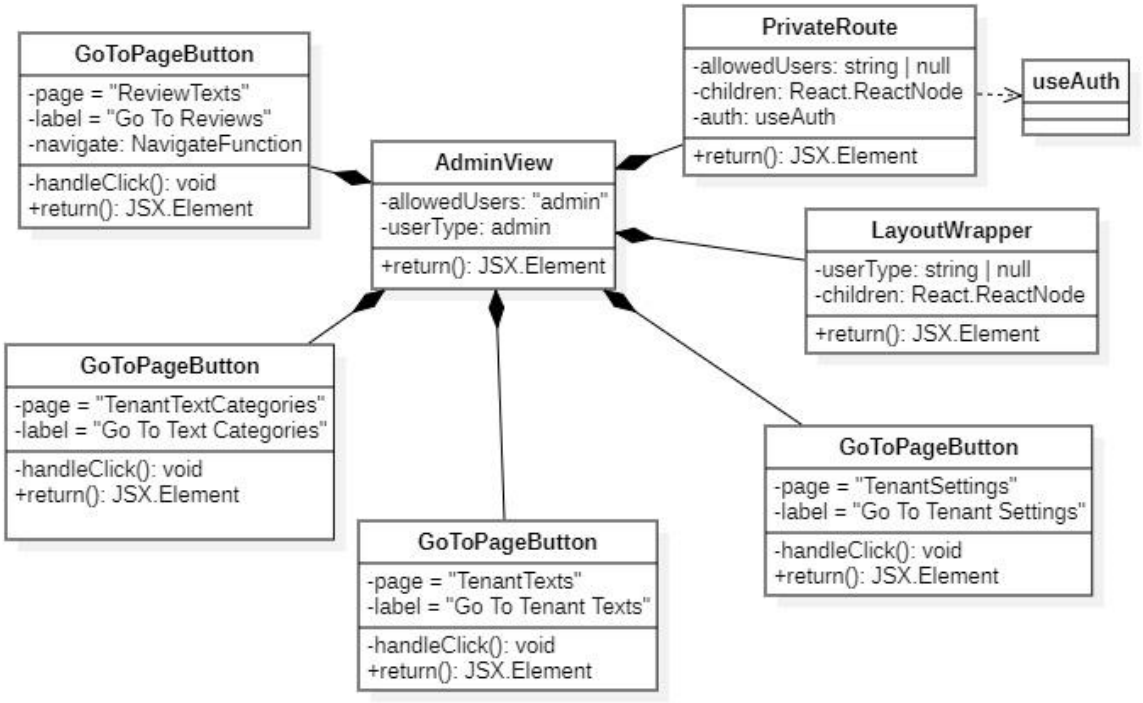

Immagine 10: Diagramma delle classi per la dashboard dell'Admin

La pagina rappresenta la vista amministrativa dell'Admin e mostra una dashboard con quattro sezioni principali, composte da un componente riutilizzabile *GoToPageButton* che consente di andare nelle pagine principali preposte alla gestione del proprio Tenant; come previsto dalle altre, la sezione è private e visibile all'utente amministratore, comprensiva di una griglia di elementi stilistici che caratterizzano la pagina sempre con l'ausilio di Material UI per impostare il tutto. Tutte le variabili private prevedono un apposito hook di gestione e la pagina ha accesso privato e controllato al solo utente Admin.

3.1.2.3.2 ReviewTextsView

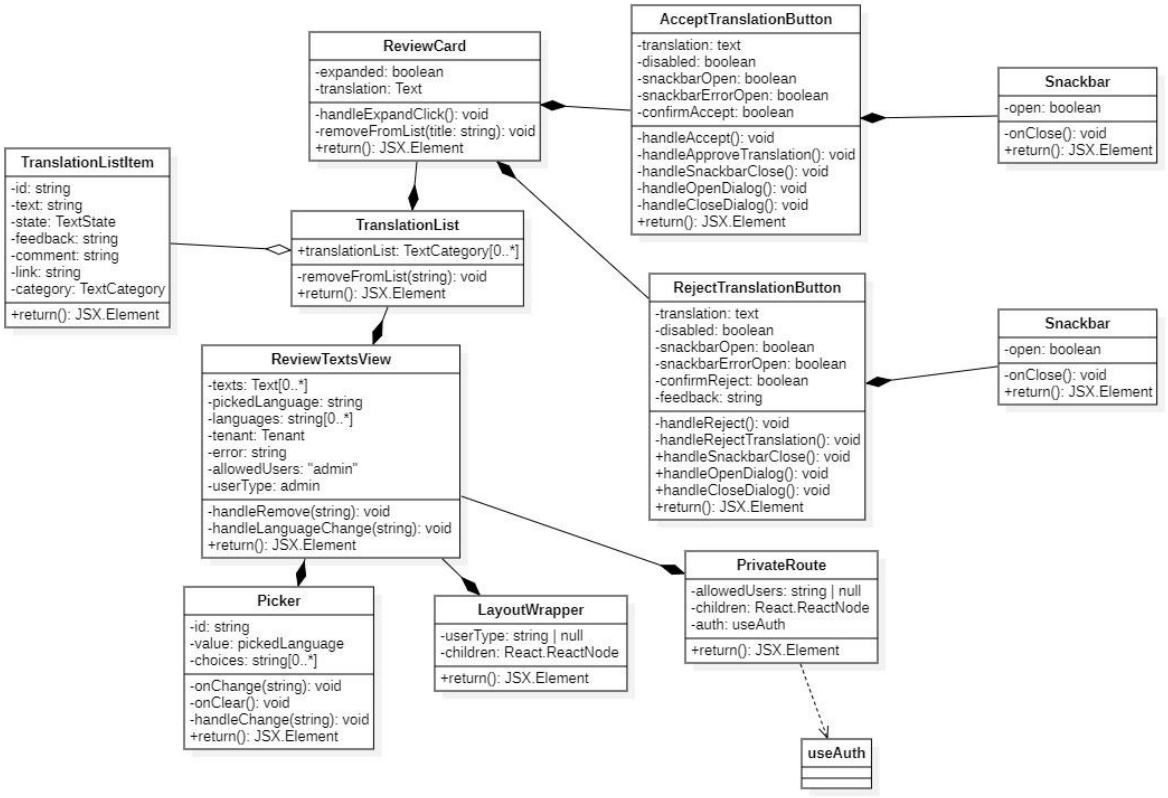

Immagine 11: Diagramma delle classi per la pagina di revisione dei testi tradotti

La pagina espone una lista di traduzioni con accesso privato e controllato al solo utente Admin da gestire, ciascuna caratterizzata dai dati di tipo *Text* come descritto inizialmente, sempre utilizzando il pattern *Compound Components* previsto da React per gli elementi di tipo lista. Tramite il *Picker* presente è possibile filtrare i testi per presenti per lingua ed è possibile accettarli o rifiutarli tramite l'utilizzo del componente *ReviewCard*, contenente due bottoni con la logica di accettazione e rifiuto e appositi componenti *Snackbar* per i messaggi di conferma e i metodi per la gestione dei relativi modali che permettono la selezione di queste opzioni. Tutte le variabili private prevedono un apposito hook di gestione.

Ciascuna di queste opzioni selezionate effettua una modifica nelle traduzioni presenti nella base di dati attraverso una chiamata API di tipo PUT.

3.1.2.3.3 CreateUserView

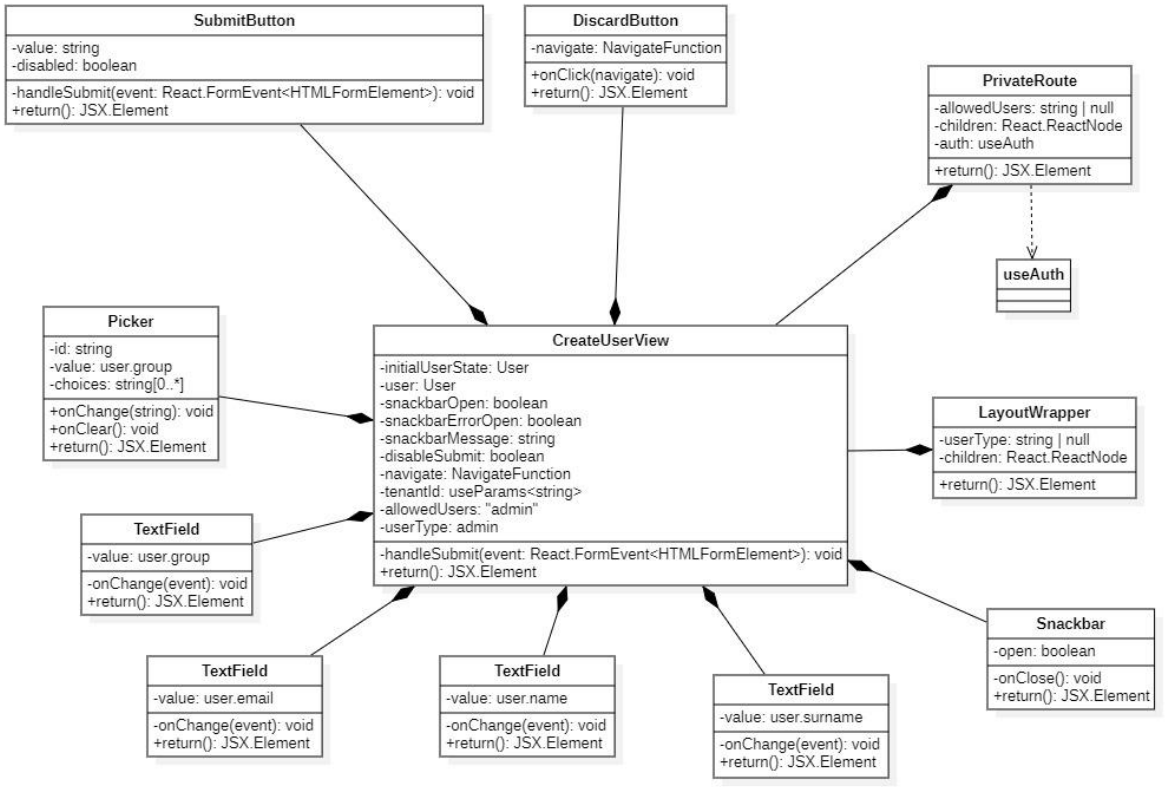

Immagine 12: Diagramma delle classi per la pagina di creazione utente

La pagina permette la creazione di un nuovo utente associato ad uno specifico Tenant con accesso privato e controllato all'utente amministratore, tramite l'invio di un form con appositi campi di tipo *TextField* forniti da Material UI. In questa è possibile associare i campi attinenti al tipo User specificato in precedenza, nello specifico come visibile nome, cognome, email e ruolo tramite il componente *Picker*.

Quando l'utente admin invia il form, viene effettuata una chiamata API per creare l'utente nel backend e aggiungerlo a un tenant (selezionato dall'utente tramite link con *useParams* o impostato automaticamente). In caso di errore, viene visualizzato un messaggio esplicativo con il componente *Snackbar*, che consente la gestione dei messaggi di conferma.

3.1.2.3.4 TenantSettingsView

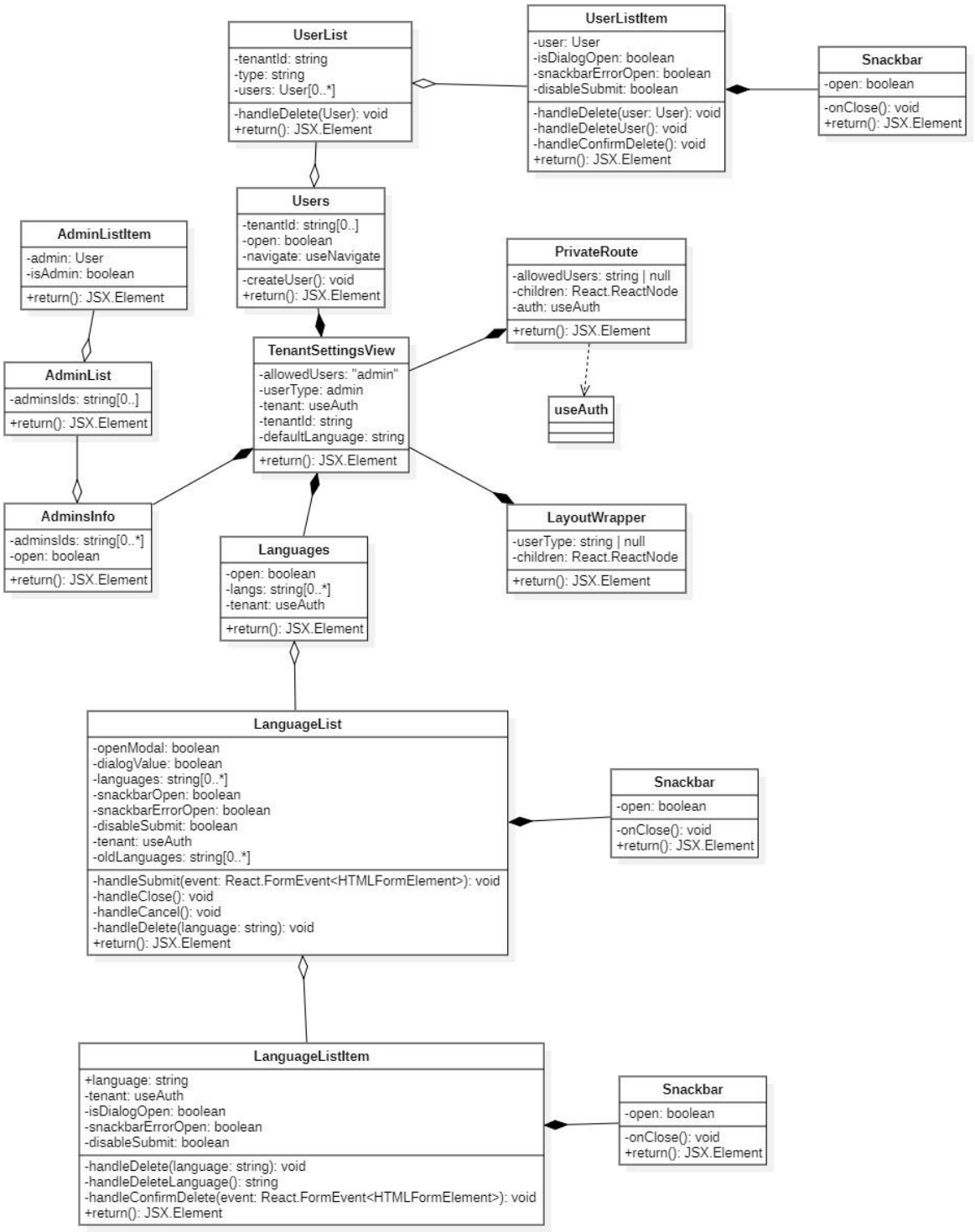

Immagine 13: Diagramma delle classi per la pagina di impostazione Tenant

La pagina rappresenta complessivamente la vista delle impostazioni del Tenant di riferimento da parte dell'utente Admin. In particolare, vengono recuperate le informazioni di un particolare Tenant tramite autenticazione e nell'insieme vengono visualizzate una serie di informazioni, comprendenti data di creazione, ID del Tenant e lingua predefinita. Inoltre, vengono definite delle componenti lista

e relativi oggetti seguendo il pattern *Compound Components* di gestione della lista di utenti e lista delle lingue presenti, con le opzioni di rimozione ed aggiunta delle stesse.

Per tali motivi, la pagina comprende diverse API, in particolare per l'operazione di aggiunta/rimozione utenti e/o lingue. Tutte le variabili private sono impostate tramite appositi hooks di riferimento e l'accesso è protetto, previsto

3.1.2.3.5 TenantTextCategoriesView

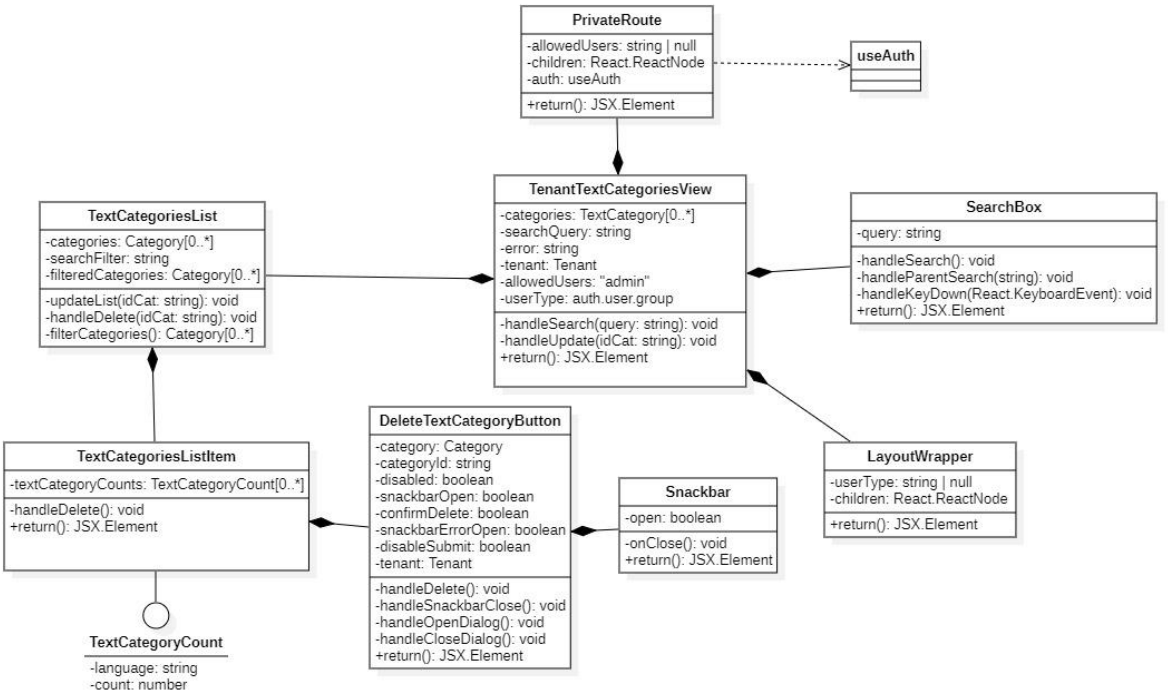

Immagine 14: Diagramma delle classi per la pagina di gestione categorie di traduzione

La pagina permette la visualizzazione all'amministratore di tutte le categorie contenute all'interno di un tenant. In particolare vengono recuperate tutte le informazioni riguardanti le categorie contenute nel tenant e vengono visualizzate una serie di informazioni, comprendenti ID della categoria, la lingua di traduzione assegnata e il numero di testi assegnati a quella categoria. In particolare in questa pagina il componente che si occupa della visualizzazione delle categorie è *TextCategoriesListItem*.
Inoltre quest'ultimo componente si occuperà anche della cancellazione di una determinata categoria, selezionata dall'amministratore, che tramite un componente interno *DeleteTextCategoryButton* e una chiamata all'API di cancellazione rimuoverà la categoria selezionata dal tenant.

3.1.2.4 Pagine SuperAdmin

L'utente SuperAdmin ,a differenza degli altri tipi di utente, avrà accesso a pagine differenti con funzionalità differenti. Lo scopo di questo utente è quello di occuparsi della creazione, inizializzazione e cancellazione di un determinato tenant.

3.1.2.4.1 SuperAdminView

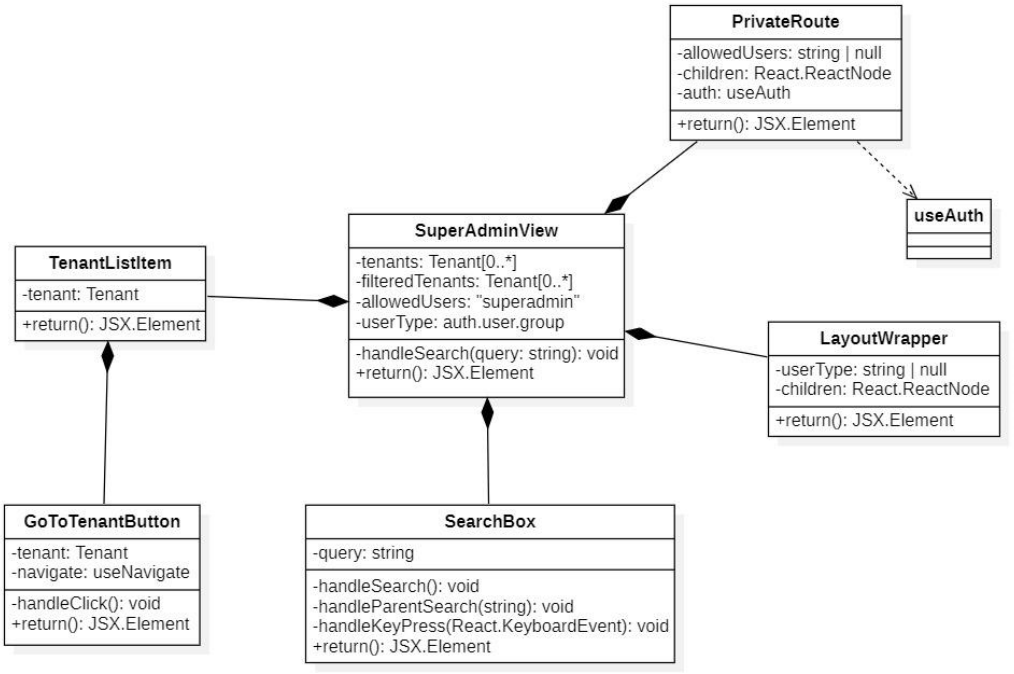

Immagine 15: Diagramma delle classi per la pagina dashboard del SuperAdmin

La pagina permette all'utente SuperAdmin di visualizzare tutti i tenant da lui creati. In particolare vengono recuperate tramite una chiamate all'API tutte le informazioni principali dei tenant e successivamente vengono visualizzate in una lista creata dal componente *TenantListItem*. Quest'ultimo componente permette la visualizzazione del nome assegnato al tenant, il suo ID, le sue lingue e tutti gli utenti che gli appartengono. Nel momento in cui il SuperAdmin voglia accedere alla pagina di gestione di un singolo tenant potrà accedere tramite il componente *GoToTenantButton*.

Nel caso il SuperAdmin conosca il nome del tenant che vuole visualizzare prioritariamente tramite il componente *SearchBox* può filtrare i risultati della vista e accedere con facilità alle informazioni desiderate.

3.1.2.4.2 SingleTenantView

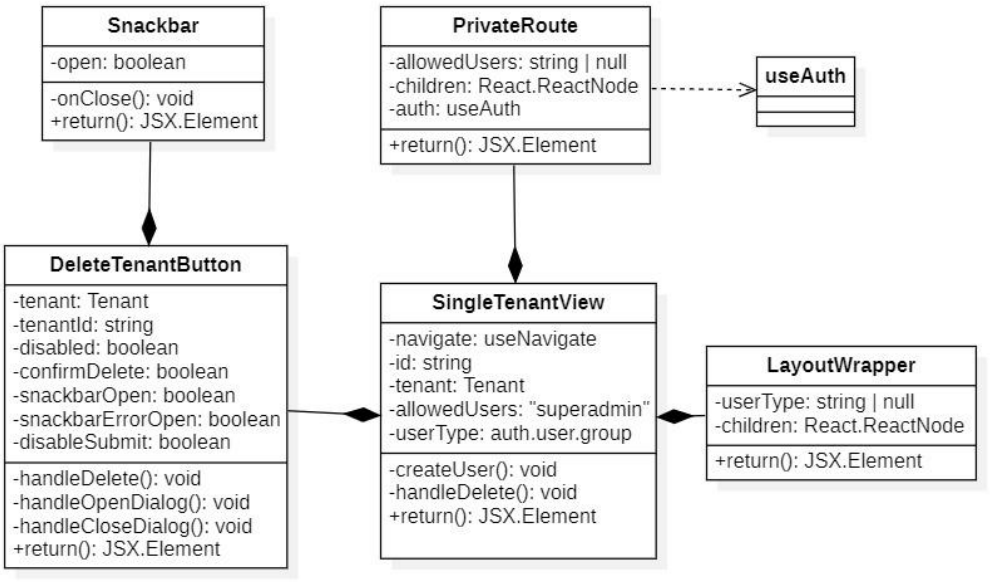

Immagine 16: Diagramma delle classi per la pagina dashboard del SuperAdmin

Questa pagina permette all'utente SuperAdmin di visualizzare in dettaglio tutte le informazioni di un singolo tenant. Vengono recuperate le informazioni del singolo tenant tramite una chiamata all'API e vengono visualizzate all'interno della pagina. Le informazioni che la pagina andrà mostrare sono il nome del tenant, la data di creazione, tutti gli utenti Admin e tutti gli utenti User. In più da questa pagina il SuperAdmin potrà creare nuovi utenti sia di tipo Admin sia di tipo User. Il componente *DeleteTenantButton* da la possibilità di eliminare il tenant visualizzato tramite una chiamata all'API

3.1.2.4.3 CreateTenantView

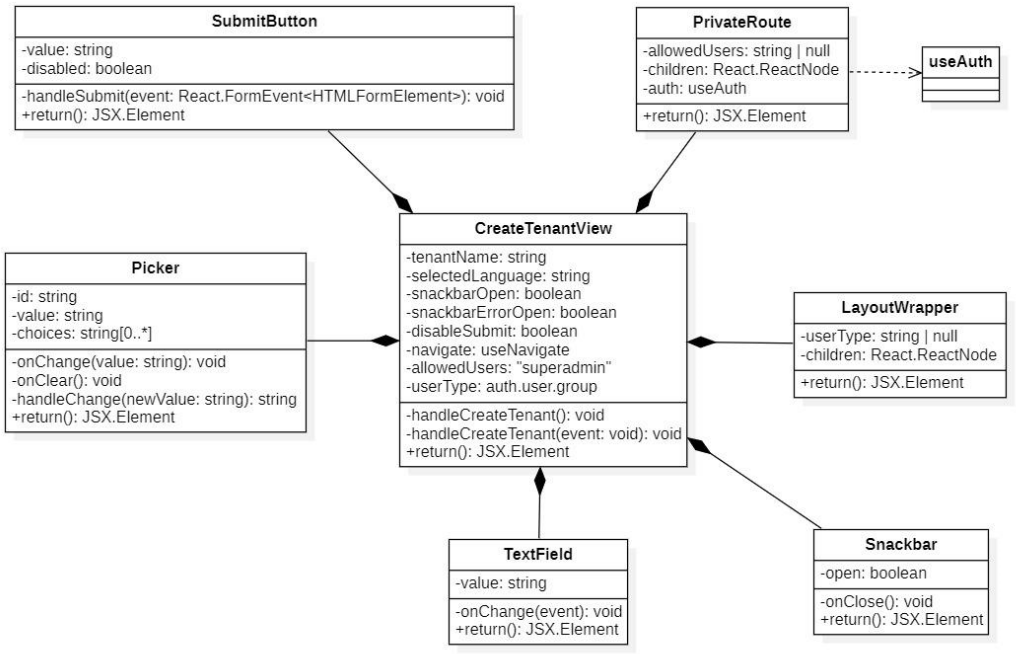

Immagine 17: Diagramma delle classi per la pagina di creazione del Tenant

La pagina permette al SuperAdmin la creazione di un nuovo tenant tramite l'inserimento di alcune informazioni. Viene richiesto tramite il componente *TextField* il nome che si vuole assegnare al tenant e la lingua di default tramite il componente *Picker*. L'interazione delle API è prevista con una chiamata di tipo POST e la conferma dell'invio delle informazioni avviene tramite il bottone *SubmitButton*. I messaggi di conferma e di errore della pagina vengono gestiti utilizzando i componenti *Snackbar* presenti nella libreria Material UI.

3.1.3 Design Pattern utilizzati

In questa sezione, sono descritti i design pattern utilizzati nell'applicazione basata su React; in particolare, sono state utilizzate diverse soluzioni, al fine di modularizzare le singole componenti e adattarle alle specifiche esigenze di realizzazione ed integrazione previste.

Di massima possiamo dettagliare l'utilizzo di:

- **Controlled Components***,* in cui i valori di input dei componenti vengono direttamente gestiti e passati dai componenti "parent" ai componenti "children" sotto forma di prop per impostare il valore dell'input e un'azione di callback per gestire le azioni eseguite sul componente "child" per aggiornare i dati contenuti nel componente "parent". Questo garantisce che i dati siano costanti tra tutti i componenti "children".

- **Presentational and Container Component**, al fine di separare la logica di presentazione dalla logica di business, realizzando componenti React che si occupano di renderizzare una vista e usarli all'interno di un componente unico.

- **React Hook**, al fine di gestire lo stato dell'applicazione in modo efficiente e visualizzare dinamicamente le informazioni, sia utilizzando gli hook già previsti in React (*useState, useEffect, useContext*), che hook personalizzati in base alle esigenze delle singole viste e componenti, come dettagliato sopra.

- **Provider**, che racchiude l'intera applicazione al fine di fornire il componente di autenticazione a tutti i componenti, semplificando la gestione dell'autenticazione e delle autorizzazioni e la sicurezza delle informazioni trasmesse. Questo garantisce di avere una singola istanza accessibile globalmente da tutte le componenti.

- **Conditional Rendering**, che consente di mostrare un contenuto diverso a seconda del ruolo dell'utente; in particolare, si prevedono dei componenti in grado di verificare il ruolo dell'utente e renderizzare opportunamente i dati utili in base al suo ruolo.
    - Questo viene utilizzato in particolare all'interno del componente *LayoutWrapper*, per renderizzare il menù adatto al tipo di utente autenticato (componente *UserMenu*).

- **Compound Components***,* al fine di modularizzare le singole componenti ideando i componenti sulla base di una gerarchia padre-figlio, in cui ciascun padre contiene uno o più componenti figlio e quindi specializzare la gestione dei dati e la personalizzazione dell'interfaccia utente in modo centralizzato, secondo una lista di opzioni unica.
    - Questo viene utilizzato per i componenti *List* e *ListItem*, presenti in varie forme come visto nei vari UML presenti sopra.

3.2 Architettura Back-end

3.2.1 Introduzione

Per il backend del servizio, è stato scelto di adottare il *Serverless Architecture Pattern*.
Questo modello di cloud computing permette di sviluppare e gestire l'applicazione senza dover gestire direttamente i server.
Il seguente pattern permette di avere un'elevata scalabilità, alta disponibilità, e bassi costi senza preoccuparsi dell'infrastruttura sottostante, grazie ad esso sarà infatti possibile scalare verso l'alto o verso il basso a seconda delle esigenze del servizio.

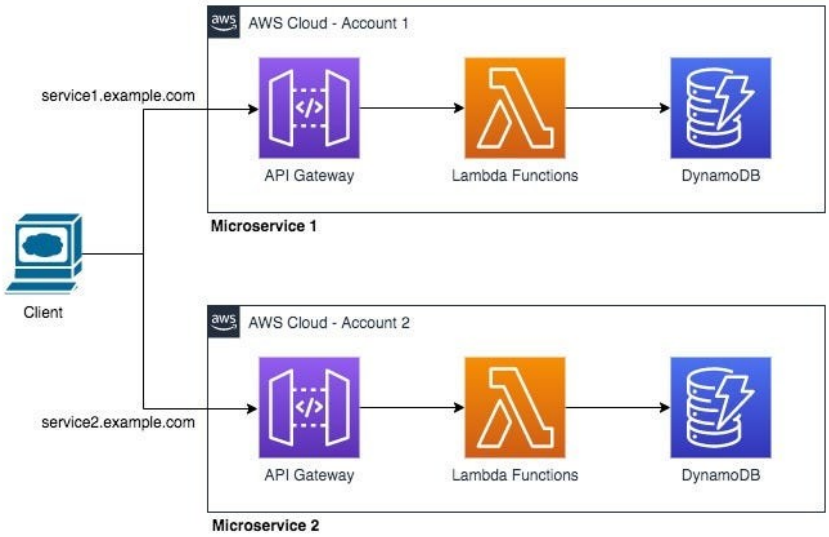

Immagine 18: Schema Serverless Architecture Pattern

Per implementare la comunicazione API, si è scelto di utilizzare il *RESTful Microservices Pattern*, che prevede l'utilizzo di API Gateway per esporre le funzioni AWS Lambda attraverso HTTPS e relative chiamate. In questo modo, i client possono comunicare in modo sincrono con i servizi di back-end tramite API RESTful. Questo consente alle chiamate di separare chiaramente i dati client-server, permettendo ai client di salvare le opportune richieste (*cacheable*), non memorizzando sul server informazioni relative al contesto (*stateless*), creando un'interfaccia di comunicazione omogenea ed estendibile a richiesta o in base alla configurazione (*code on demand*).

Possiamo dettagliare le singole componenti del pattern in questo modo:

- API Gateway, utilizzato per gestire l'autorizzazione, autenticazione, instradamento e relativo versionamento delle singole API. Esso interagisce con Cognito in fase di autenticazione, effettuando le chiamate necessarie in base al tipo di utente come sopra descritto;
- DynamoDB, utilizzato per la persistenza dei singoli dati, permettendo la configurazione serverless quindi gestita in modo scalabile e flessibile grazie alla sua natura non relazionale;
- le funzioni Lambda, utilizzate per implementare le *CRUD Endpoints,* API che rappresentano le operazioni fondamentali previste dal sistema (Create-Read-Update-Delete), offrendo un'opportuna interfaccia standardizzata per esigenza.

In questo modo, la comunicazione tra le diverse parti del sistema avviene in modo sincrono, permettendo al client di inviare una richiesta e ricevere una risposta dal server recuperando i relativi dati nello stesso momento. Anche lato back-end, questa scelta permette di creare un'architettura modulare e distribuita, che può essere facilmente scalata e mantenuta.

3.2.2 Schema base di dati

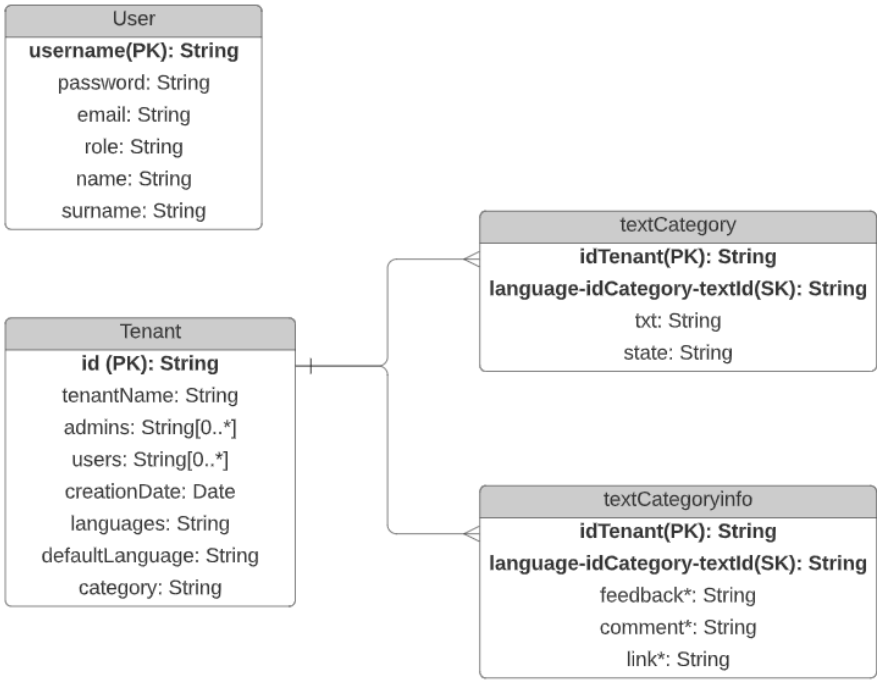

Immagine 19: Schema base di dati

In questa sezione, viene presentato lo schema base di dati realizzato con DynamoDB relativo all'architettura back-end del servizio descritto. Le informazioni sono separate rispetto alla parte front-end visualizzabile dall'utente, tale da organizzare i dati in modo tale da mantenere distinti i dati utili per il recupero dei dati lato back-end e i dati usati in fase di traduzione. In questo modo è stata garantita una maggiore separazione delle responsabilità tra le varie componenti del sistema.

Possiamo dettagliare questa composizione (dove presente il simbolo di asterisco [*] si indica la presenza di possibili valori nulli):

- **Tenant**, rappresentante i singoli Tenant, comprensiva di:
  - un identificativo univoco;
  - il nome;
  - la lista degli Admin presenti;
  - la lista degli User presenti;
  - la data di creazione;
  - le lingue presenti;
  - la lingua predefinita;
  - la categoria di riferimento.

- **textCategory**, rappresentante i dati di un singolo testo in una lingua con stato e feedback, comprensiva di:
    - identificativo del Tenant di riferimento;
    - la lingua associata;
    - il testo;
    - il relativo stato.
- **textCategoryInfo**, rappresentante i metadati appartenenti ad un testo indipendentemente dalla lingua, comprensiva di:
    - identificativo del Tenant di riferimento;
    - identificativo della categoria
    - commento al testo;
    - feedback del testo;
    - link di riferimento all'esterno per risorse aggiuntive sulla traduzione.
- **User**, rappresentante i dati di uno specifico utente, salvata all'interno di AWS Cognito e usando tali dati in fase di autenticazione, comprensiva di:
    - username (chiave primaria);
    - password;
    - email di riferimento;
    - ruolo dell'utente;
    - nome;
    - cognome.

La scelta di utilizzare un ID testo univoco fra tutti i testi anziché solo all'interno di una categoria ha semplificato l'uso dei testi da parte del sito web, senza la necessità di specificare la categoria di appartenenza.

La combinazione delle chiavi primarie può a prima vista comportare alcune difficoltà, per esempio sapendo che un testo può far parte di più categorie, rendendo così necessari maggiori controlli in fase implementativa (situazione limitata, che si verifica nel caso in cui vengano applicati molteplici filtri). Tuttavia, la velocità di interazione garantisce che, con una sola chiamata al database, si abbia un recupero dei dati in tempi rapidi, dato che un particolare testo può essere presente in più categorie di traduzione per le varie lingue; l'iterazione è ripetuta per il solo identificativo dei testi e la maggiore ridondanza comporta comunque un'interazione veloce e valida in fase di lettura.

3.2.3 Design Pattern utilizzati

Nella seguente sezione, vengono descritti i design pattern adottati lato backend, vista la descrizione data inizialmente:

- **Function-as-a-Service (FaaS):** consente di scrivere funzioni senza la necessità di gestire l'infrastruttura sottostante. Questo ha permesso di concentrarsi sullo sviluppo delle singole funzionalità senza preoccuparsi della configurazione o gestione dei server;
- **Backend-as-a-Service (BaaS):** consente di utilizzare servizi cloud per gestire funzionalità comuni del backend, come l'autenticazione degli utenti, lo storage dei dati e la gestione delle notifiche push. Ciò ha permesso di ridurre i tempi di sviluppo e semplificare la gestione delle funzionalità del backend;

- **Event-driven computing:** consente di scrivere codice che viene eseguito in risposta a eventi specifici, come ad esempio una richiesta API. CIò ha consentito la creazione di un'architettura scalabile e reattiva, in grado di gestire carichi di lavoro elevati senza interruzioni;
- **API Gateway:** consente di creare un punto di ingresso centralizzato per l'accesso alle funzioni serverless. Il suo utilizzo ha semplificato la gestione delle API e delle relative autorizzazioni e autenticazioni.

3.2.4 Documentazione API

La seguente sezione descrive ad alto livello la libreria creata dal gruppo *SWEG* per comunicare con l'applicazione, fornendo una panoramica di massima delle API[G] disponibili, comprensiva di breve descrizione delle operazioni disponibili e dei dati che possono essere acceduti.

La loro strutturazione dettagliata è presente nel documento *Manuale Sviluppatore v.1.0.0*; in questo modo, gli sviluppatori interessati ad estendere le funzionalità avranno una piena comprensione del software, implementando nuove funzionalità in modo programmatico ed automatizzato senza dover accedere manualmente all'interfaccia utente.

La sezione qui presente, pertanto, offre una visione di massima e con scopo illustrativo. Tutte le chiamate sono divise, nelle rispettive sottosezioni, in 3 gruppi principali: Tenant, User e Text. Ciascuna sarà caratterizzata da un parametro, convenzionalmente indicato con parentesi graffe. La chiamata API, relativo endpoint ed esempi di chiamata sono realizzati estesamente nel documento citato nel capitolo §6 e nelle relative sottosezioni.

3.2.4.1 Chiamate GET

3.2.4.1.1 Tenant

- **/tenant/allTenants:**
  Ritorna la lista di tutti i Tenant
- **/tenant/{TenantId}/info:**
  Ritorna le informazioni relative ad un Tenant specifico
- **/tenant/{TenantId}/admins:**
  Ritorna la lista degli admin associati al Tenant
- **/tenant/{TenantId}/users:**
  Ritorna la lista degli utenti associati al Tenant
- **/tenant/{TenantId}/secondaryLanguage:**
  Ritorna la lista delle lingue associate al Tenant
- **/tenant/{TenantId}/allCategories:**
  Ritorna la lista delle categorie presenti nel Tenant
- **/tenant/{TenantId}/countTexts:**
  Ritorna il numero di testi presenti nel Tenant, suddivisi per lingua

3.2.4.1.2 User

- **/user/{Username}/getResetCode:**
  Invoca la procedura di reset password

- **/user/getUsers:**
  Ritorna la lista degli Utenti registrati nel sistema
- **/user/{Username}/adminGetUser:**
  Ritorna le informazioni relative ad un utente specifico
- **/user/{Username}/tenant:**
  Ritorna il Tenant associato all'utente

3.2.4.1.3 Text

- **/text/{TenantId}/allTexts**
  Ritorna la lista di tutti i testi associati al Tenant
- **/text/{TenantId}/{Language}/{Category}/{Title}/text:**
  Ritorna le informazioni relative ad un testo specifico
- **/text/{TenantId}/category/{Category}/{Title}/translationLanguages:**
  Ritorna la lista delle lingue tradotte/da tradurre di un testo specifico

3.2.4.2 Chiamate PUT

3.2.4.2.1 Text

- **/text/{TenantId}/category/{Category}/{Title}/originalText:**
  Aggiorna le informazioni relative ad un testo
- **/text/{TenantId}/{Language}/{Category}/{Title}/translation:**
  Aggiorna la traduzione di un testo
- **/text/{TenantId}/{Language}/{Category}/{Title}/approveTranslation:**
  Approva la traduzione di un testo
- **/text/{TenantId}/{Language}/{Category}/{Title}/rejectTranslation:**
  Rifiuta la traduzione di un testo

3.2.4.3 Chiamate POST

3.2.4.3.1 Tenant

- **/tenant/create:**
  Crea un Tenant
- **/tenant/{TenantId}/addLanguage:**
  Aggiunge lingue ad un Tenant

3.2.4.3.2 User

- **/user/create/{TenantId}:**
  Crea un utente associato ad un Tenant
- **/user/resetPassword:**
  Procedura di reset password utente

3.2.4.3.3 Text

- **/text/{TenantId}/originalText:**
  Aggiunge un testo al Tenant

3.2.4.4 Chiamate DELETE

3.2.4.4.1 Tenant

- **/tenant/{TenantId}:**
  Elimina un Tenant
- **/tenant/{TenantId}/language/{Language}:**
  Elimina una lingua dal Tenant e rispettive traduzioni
- **/tenant/{TenantId}/{Category}/category:**
  Elimina una categoria dal Tenant e rispettivi testi

3.2.4.4.2 User

- **/user/{Username}/delete:**
  Elimina un utente

3.2.4.4.3 Text

- **/text/{TenantId}/category/{Category}/{Title}/originalText:**
  Elimina un testo dal Tenant

4 Requisiti soddisfatti

La successiva sezione si pone l'obiettivo di mostrare, secondo quanto riportato dal documento *Piano di Progetto v.2.0.0*, la soddisfazione dei singoli requisiti funzionali presenti, in base al tipo previsto e opportunamente classificato sotto.

4.1 Tabella requisiti soddisfatti

Si intende che la numerologia di ciascuno rispecchi ove previsto gli UC presenti, mentre le sigle sotto riportate indicano:

- ROF - Requisito Obbligatorio Funzionale;
- RDF - Requisito Desiderabile Funzionale.

Rispetto alla stessa tabella del documento *Analisi dei Requisiti v.2.0.0*, si ha un'apposita colonna *Stato*, che indica la soddisfazione del tipo di requisito.

Codice	***Descrizione***	***Stato***
ROF 1	L'utente deve poter accedere all'applicazione	Soddisfatto
RDF 2	L'utente deve poter recuperare la propria password	Soddisfatto
RDF 3	Il sistema deve visualizzare un messaggio di errore esplicativo relativamente al mancato accesso	Soddisfatto
ROF 4	L'utente deve poter uscire dalla propria sessione	Soddisfatto
ROF 5	Il SuperAdmin deve poter creare un Tenant e associargli le opportune informazioni	Soddisfatto

ROF 6	Il SuperAdmin deve poter cancellare il Tenant che ha creato	Soddisfatto
ROF 7	Il SuperAdmin deve poter visualizzare la lista dei Tenant disponibili e i singoli Tenant	Soddisfatto
ROF 8	Il SuperAdmin deve poter visualizzare il singolo Tenant in lista	Soddisfatto
ROF 9	Il SuperAdmin deve poter scegliere la lingua di default del Tenant	Soddisfatto
ROF 10	L'utente deve poter visualizzare le lingue secondarie presenti e i dettagli della singola lingua	Soddisfatto
RDF 11	Gli utenti amministratori devono poter visualizzare tutte le lingue presenti nel Tenant, per poterle eventualmente modificare	Soddisfatto
ROF 12	L'Admin deve poter cancellare la lingua secondaria del Tenant	Soddisfatto
ROF 13	L'Admin deve poter creare un utente associato ad un Tenant, associandogli una serie di informazioni	Soddisfatto
ROF 14	L'Admin deve poter visualizzare gli utenti attivi per il proprio Tenant, visualizzando i dettagli del singolo utente	Soddisfatto
ROF 15	L'Admin deve poter cancellare un utente attivo nel proprio Tenant	Soddisfatto
ROF 16	L'Admin deve poter creare un testo originale da inserire nel proprio Tenant, associandogli una serie di informazioni	Soddisfatto
ROF 17	L'Admin deve poter visualizzare la lista dei testi originali e i dettagli di un singolo testo	Soddisfatto
ROF 18	L'Admin deve poter modificare un testo originale presente	Soddisfatto
ROF 19	L'Admin deve poter cancellare un testo originale presente	Soddisfatto
ROF 20	L'Admin deve visualizzare la lista delle traduzioni da approvare ed il dettaglio di una delle traduzioni presenti	Soddisfatto
ROF 21	L'Admin deve poter approvare i testi presenti	Soddisfatto
ROF 22	L'Admin deve poter rifiutare i testi presenti	Soddisfatto
ROF 23	L'Admin deve poter creare una categoria di traduzioni, associando ad essa una serie di informazioni	Soddisfatto
ROF 24	L'utente deve poter visualizzare le categorie di traduzioni presenti	Soddisfatto
ROF 25	L'Admin deve poter modificare una categoria di traduzione presente	Soddisfatto
ROF 26	L'Admin deve poter cancellare una categoria di traduzione presente	Soddisfatto

ROF 27	L’utente deve poter visualizzare le traduzioni non eseguite presenti e il dettaglio di una singola selezionata	Soddisfatto
ROF 28	L’utente deve poter visualizzare le traduzioni già eseguite presenti e il dettaglio di una singola selezionata	Soddisfatto
ROF 29	Gli User devono poter inserire delle traduzioni	Soddisfatto
RDF 30	L’utente avrà a disposizione una funzione di ricerca testi	Soddisfatto
RDF 31	L’utente avrà a disposizione una funzione di filtro testi	Soddisfatto
RDF 32	L’Admin avrà a disposizione un’opzione di modifica della lingua di default	Soddisfatto
RDF 33	Il sistema permetterà un’integrazione da parte delle webapp tramite le API	Soddisfatto
RDF 34	Il sistema permetterà il recupero dati tramite API	Soddisfatto
RDF 35	Il sistema notifica all’utente la cancellazione di una traduzione a lui associata	Soddisfatto
RDF 36	Il sistema invierà un feedback sulla valutazione della traduzione presente all’utente una volta che l’Admin ha approvato o rifiutato	Soddisfatto
RDF 37	Il sistema permette l’esportazione delle traduzioni in un formato specifico	Soddisfatto
RDF 38	L’Admin ha la possibilità di modificare i dati dei propri utenti	Non soddisfatto
RDF 39	L’utente ha la possibilità di condividere le proprie traduzioni	Non soddisfatto
RDF 40	Gli utenti amministratori possono avere degli strumenti per generare report sulla gestione dei Tenant, sulle attività degli utenti e sullo stato delle traduzioni all'interno di un Tenant.	Non soddisfatto
RDF 41	Gli utenti possono visualizzare un’anteprima del testo tradotto che stanno creando	Non soddisfatto
RDF 42	Gli Admin possono creare e gestire glossari specifici per il loro Tenant in modo da migliorare la coerenza delle traduzioni e ridurre il tempo necessario per tradurre un testo	Non soddisfatto
RDF 43	Il sistema offre la possibilità di integrare un sistema di traduzione automatica che consenta agli utenti di tradurre rapidamente il testo utilizzando tecnologie di traduzione automatica, ma con possibilità di revisione da parte degli utenti.	Non soddisfatto
ROF 44	L’Admin può filtrare opportunamente i testi del proprio Tenant sulla base di varie opzioni	Soddisfatto
ROF 44.1	L’Admin può filtrare i testi del proprio Tenant sulla base della categoria di traduzione	Soddisfatto

ROF 44.2	L'Admin può filtrare i testi del proprio Tenant sulla base della lingua di traduzione	Soddisfatto
ROF 44.3	L'Admin può filtrare i testi del proprio Tenant sulla base del loro stato di traduzione	Soddisfatto
ROF 44.4	L'Admin può filtrare i testi del proprio Tenant sulla base del loro identificativo	Soddisfatto
ROF 45	Il SuperAdmin può ricercare i propri Tenant sulla base di varie opzioni	Soddisfatto
ROF 45.1	Il SuperAdmin può ricercare i propri Tenant per categoria di traduzione	Soddisfatto
ROF 45.2	Il SuperAdmin può ricercare i propri Tenant per nome	Soddisfatto
ROF 45.3	Il SuperAdmin può ricercare i propri Tenant per identificativo	Soddisfatto
ROF 46	L'utente può filtrare i testi da tradurre sulla base di varie opzioni	Soddisfatto
ROF 46.1	L'utente può filtrare i testi da tradurre sulla base del loro identificativo	Soddisfatto
ROF 46.2	L'utente può filtrare i testi da tradurre sulla base di una lingua di traduzione	Soddisfatto
ROF 46.3	L'utente può filtrare i testi da tradurre sulla base del loro stato di traduzione	Soddisfatto

4.2 Grafici requisiti soddisfatti

In merito alla soddisfazione dei vari requisiti funzionali, il gruppo *SWEG* ha soddisfatto 50 su 56 requisiti, arrivando ad una copertura dell'89%.

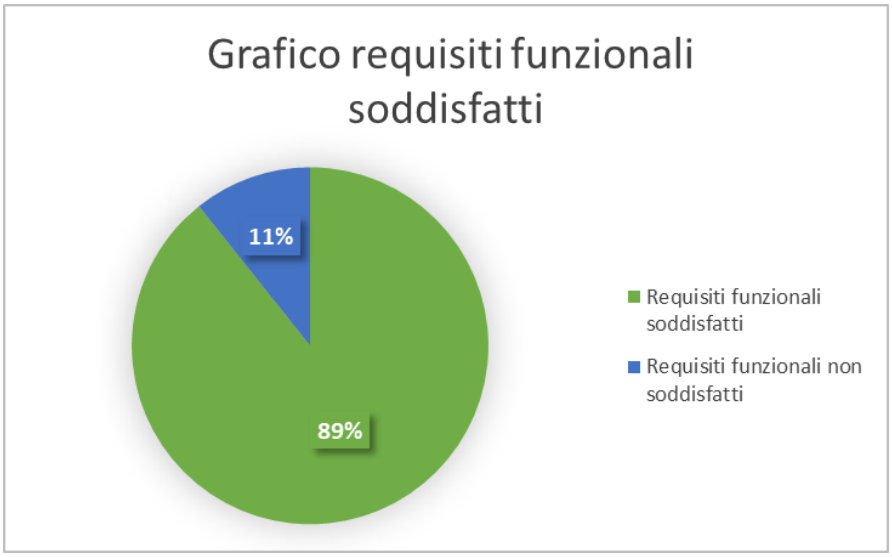

Grafico 1 - Grafico requisiti funzionali soddisfatti

Per quanto riguarda invece la copertura dei requisiti obbligatori, la copertura rilevata è del 100%.

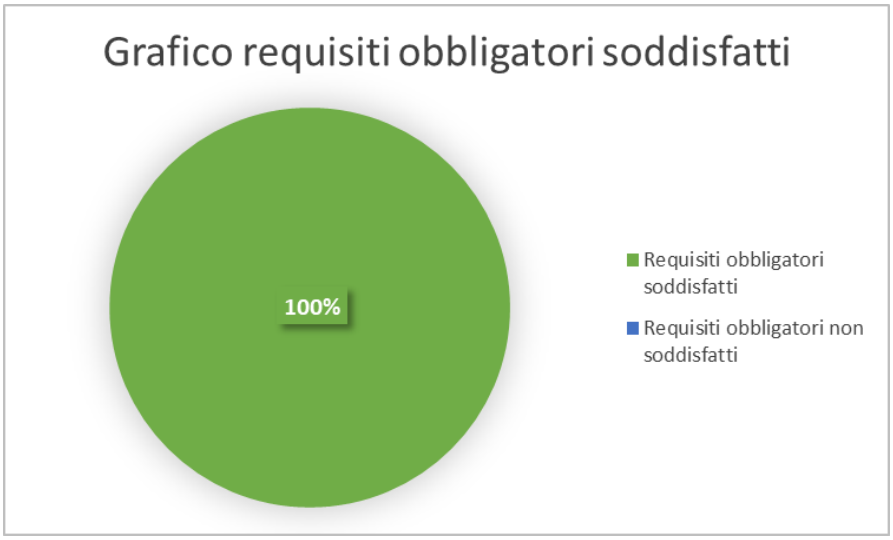

Grafico 2 - Grafico requisiti obbligatori soddisfatti