Specifica tecnica
Progetto Piattaforma di Localizzazione Testi

submarines.g4@gmail.com

Informazioni sul documento

Responsabile	Niccolò Fasolo
Redattori	Samuel Scarabottolo
	Niccolò Fasolo
	Michael Amista'
Verificatori	Michael Amista'
	Andrea Longo
Uso	Esterno
Destinatari	Prof. Tullio Vardanega
	Prof. Riccardo Cardin
	Michele Massaro - Zero12 s.r.l.
Versione	1.0.0

Sommario

Questo documento racchiude tutte le scelte architetturali che il gruppo ha preso per la realizzazione del capitolato. Per la descrizione del prodotto sono stati utilizzati i diagrammi delle classi e di sequenza

Registro delle Modifiche

Versione	Data	Autore	Ruolo	Descrizione
1.0.0	2023/06/01	Niccolò Fasolo	*Responsabile*	Approvazione e rilascio del documento
0.4.0	2023/06/01	Andrea Longo	*Verificatore*	Verifica complessiva del documento
0.3.1	2023/05/31	Michael Amista', Niccolò Fasolo	*Programmatore*	Inseriti i diagrammi per la parte front-end
0.3.0	2023/05/05	Michael Amista'	*Verificatore*	Verifica di quanto scritto.
0.2.1	2023/05/03	Samuel Scarabottolo	*Progettista*	Stesura §2.2.1
0.2.0	2023/04/28	Michael Amista'	*Verificatore*	Verifica dei capitoli scritti.
0.1.1	2023/04/24	Samuel Scarabottolo	*Progettista*	Stesura di §2.1 e §2.2.1
0.1.0	2023/04/22	Michael Amista'	*Verificatore*	Verifica di quanto fatto.
0.0.2	2023/04/20	Samuel Scarabottolo	*Progettista*	Stesura di §1 (con annessi sottocapitoli)
0.0.1	2023/04/15	Samuel Scarabottolo	*Progettista*	Creato prototipo specifica architetturale

Indice

Elenco delle tabelle

Elenco delle figure

1 Introduzione

1.1 Scopo del documento

Questo documento ha lo scopo di descrivere e motivare tutte le scelte architetturali adottate dal team in fase di progettazione e codifica del prodotto. Vengono riportati dunque i diagrammi delle classi per descrivere l'architettura e le funzionalità principali. Alla fine del documento vi è presente una sezione dedicata a tutti i requisiti che il gruppo è riuscito a soddisfare, così da avere un'ampia visione sullo stato di avanzamento del prodotto.

1.2 Scopo del prodotto

Lo scopo di Tean Submarines e di Zero12 è la creazione di una piattaforma in grado di gestire i testi delle localizzazioni di mobile apps e webapps.
Il sistema, gestito in modalità multi-tenant, sarà costituito principalmente da un'API^{G} che permette agli sviluppatori di accedere alle traduzioni dei loro testi da inserire all'interno delle apps, e da una webapp di backoffice (CMS) che permette agli amministratori del sistema di accedere al database di traduzioni.

1.3 Glossario

Per chiarezza c'è un documento "Glossario v.1.0.0" presente nella documentazione che va a chiarire tutti i termini e le espressioni che possono risultare ambigue. Questi termini avranno il pedice 'G', esempio "parola$_{G}$". In caso venga citato un documento verrà inserito il pedice 'D', esempio "documento$_{D}$".

1.4 Riferimenti

1.4.1 Riferimenti normativi

- *Capitolato d'appalto C4 - Piattaforma di localizzaziione dei testi per app e webapp.*;
  https://www.math.unipd.it/~tullio/IS-1/2022/Progetto/C4.pdf
- *AnalisiDeiRequisiti-v2.0.0*

1.4.2 Riferimenti informativi

- *Regolamento del progetto didattico:*
  https://www.math.unipd.it/~tullio/IS-1/2022/Dispense/PD02.pdf
- *Slide P2 del corso di ingegneria del software - Diagrammi delle classi (UML):*
  https://www.math.unipd.it/~rcardin/swea/2023/Diagrammi%20delle%20Classi.pdf
- *Model-View Patterns - Materiale didattico del corso di Ingegneria del Software:*
  https://www.math.unipd.it/~rcardin/sweb/2022/L02.pdf

2 Architettura del prodotto

2.1 Architettura generale

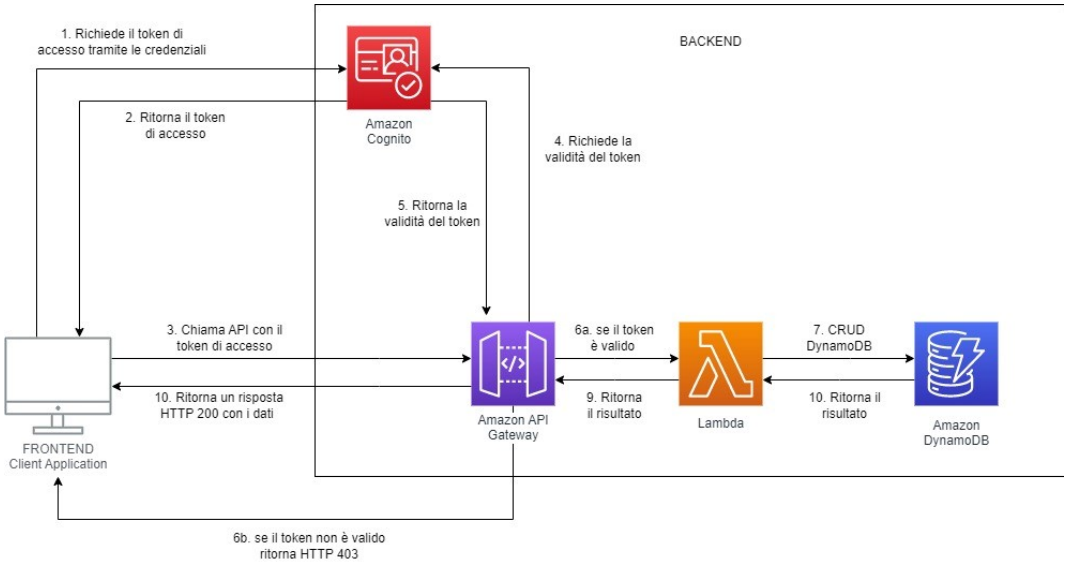

Figura 1: Schema generale del sistema

Come da vincolo progettuale dal capitolato si è deciso di utilizzare un'architettura a microservizi, i quali comunicano con il Frontend. In particolare sono stati individuati i seguenti microservizi:

- Amazon Cognito: fornisce autenticazione, autorizzazione e gestione degli utenti per le web app e dispositivi mobili.
- API Gateway: servizio che semplifica la creazione, pubblicazione, manutenzione, monitoraggio e protezione delle API. Quest'ultime consentono l'accesso delle applicazioni ai dati, alla logica aziendale o alle funzionalità dai servizi backend. Nel dettaglio API Gateway consente di creare API RESTful e WebSocket che consentono la comunicazione bidirezionale delle applicazioni, tutto in tempo reale.
- AWS Lambda: consente di eseguire codice senza che l'utente debba effetturare il provisioning o gestica le infrastrutture
- Amazon DynamoDB: servizio di database NoSQL per la gestione per l'appunto di un database. Per questo prodotto si è deciso di strutturare la base di dati attraverso un *List Access Pattern*, quest'ultima permette l'accesso a diverse funzionalità solo ad alcune categorie di utenti.

2.2 Back-end

2.2.1 Descrizione e schema

Di seguito forniamo uno schema rappresentativo del lato back-end dell'applicativo.

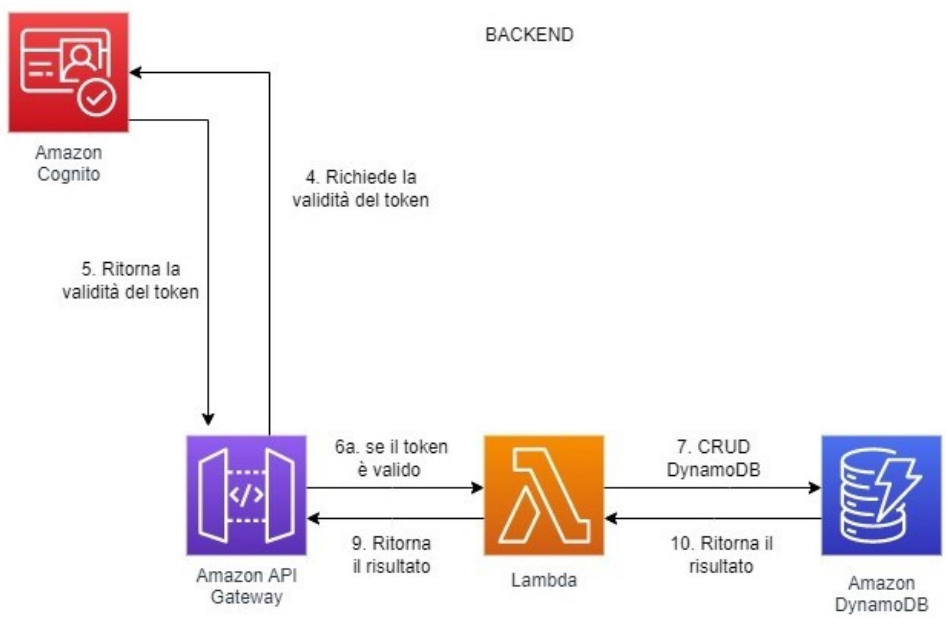

Figura 2: Schema della parte back-end

Per prima cosa, l'utente inserisce le credenziali fornite nel form di login. Nel caso le credenziali siano corrette, il sistema restituisce un token d'accesso. Viene successivamente effettuata una chiamata tramite API Gateway ad Amazon Cognito, che verifica la validità del token ricevuto. Se la verifica va a buon fine, l'utente può usufruire dell'intero sistema, interagendo con AWS Lambda e DynamoDB eseguendo codice ed query al database, altrimenti viene visualizzato un errore.

2.2.2 Progettazione database (DynamoDB)

Per la progettazione della base di dati abbiamo seguito la guida DynamoDB fornita da AWS. Per iniziare a progettare una tabella DynamoDB in grado di dimensionare in maniera corretta, è necessario seguire delle procedure per l'identificazione dei modelli di accesso richiesti dai sistemi di operations e business support. Di seguito elenchiamo i metodi che abbiamo individuato:

- Accedere a tutte le traduzioni del tenant (a cui le traduzioni appartengono): key e la lingua di traduzione default;
- Accedere ad una traduzione in dettaglio(vedere tutte le lingue, chi ha modificato, quando);
- Accedere al versionamento di una traduzione(vedere lista delle 5 vecchie versioni);
- Accedere al versionamento di una traduzione in dettaglio (vedere tutte le lingue, da chi e quando è stata modificata);
- Accedere alle informazioni del tenant;
- Accedere a tutti gli utenti presenti in un tenant;
- Accedere ad una lista completa di tutti i tenant;
- *(FILTRAGGIO)* accedere a tutte le traduzioni che contengono la chiave inserita;
- *(FILTRAGGIO)* accedere a tutte le traduzioni con una certa data di creazione;

- *(FILTRAGGIO)* accedere a tutte le traduzioni che sono state pubblicate e non;
- Creazione di una nuova traduzione;
- Inserimento delle traduzioni nelle lingue disponibili;
- Modifica di una traduzione;
- Pubblicazione di una traduzione;
- Eliminazione di un tenant;
- Eliminazione di una traduzione

In base alla lista di accesso, sono sate decise le PK (partition key) e SK (sort key) in modo da facilitare l'esecuzione delle query. Come PK vi è un ID univoco per l'identificazione del tenant, come SK invece c'è una stringa che identifica 3 diverse categorie di elementi:

- *TENANT#<TenantID>*: identifica un singolo tenant;
- *TRAD#<KeyTraduzione>*: identifica una singola traduzione;
- *USER#<UserID>*: identifica un singolo utente.

Questo utilizzo di stringhe del tipo *Tipo#<ID>*, oltre che ad identificare facilmente cosa è ogni riga della tabella, permette di visualizzare facilmente la tabella stessa. In quanto le tuple vengono ordinate secondo la SK.
Inoltre negli attributi sono stati inseriti tutti i dati necessari per "rispondere" alle richieste presenti nel List Access Pattern.
Sono stati scelti i seguenti attributi:

- Tupla di tipo *TENANT#<tenantId>*:
  - tenantName: stringa contenente il nome del tenant;
  - defaultTranslationLanguage: definisce la lingua della traduzione di default che verrà utilizzata per quel tenant;
  - listAvaiableLanguages: lista di stringhe che definisce tutte le lingue in cui è possibile tradurre nel tenant;
  - numberTranslationAviable: numero che specifica quante traduzioni ha a disposizione un tenant;
  - token: stringa che indica il token del tenant;
  - listUserTenant: lista che elenca tutti i vari utenti del tenant.
- Tupla di tipo *TRAD#<translationKey>*:
  - modifiedbyUser: stringa che identifica l'ultimo utente che modificato la traduzione;
  - modificationDate: data che indica l'ultima modifica della traduzione;
  - creationDate: data che indica quando è stata creata la traduzione;
  - published: valore booleano che indica se la traduzione è stata pubblicata o no;
  - defaultTranslationinLanguage: mappa con la traduzione nelle varie lingue;
  - versionedTranslations: array di mappe, ognuna delle quali corrisponde ad una delle ultime 5 versioni della traduzione, inclusa quella corrente.

- Tupla di tipo *USER#<UserID>*:
  - userEmail: stringa che indica l'indirizzo email dell'utente;
  - userCreationDate: data che indica quando è stato creato l'account;
  - username: stringa che indica l'username dell'utente;
  - name: stringa che indica il nome dell'utente;
  - lastName: stringa che indica il cognome dell'utente;
  - role: stringa che indica il ruolo dell'utente.

2.2.3 Schema database

L'immagine che segue è stata utilizzata solo a scopo progettuale, non rappresenta dunque ciò che veramente è presente su DynamoDB.

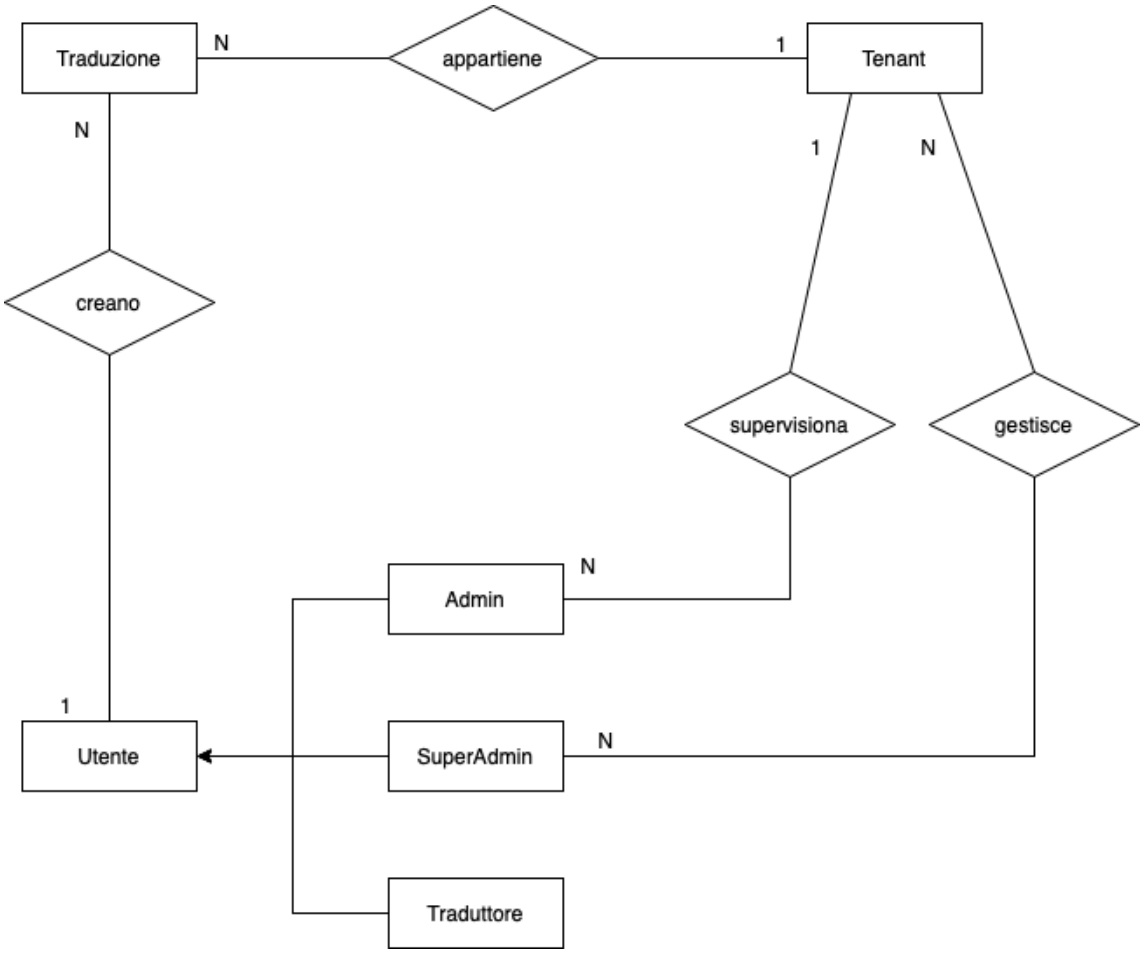

Figura 3: Schema del database

Come si può notare gli utenti si suddividono in 3 tipologie:

- *Admin*: il quale ha il potere di supervisionare i tenant;
- *SuperAdmin*: il quale può creare, eliminare e gestire tutto ciò che riguarda il tenant stesso;
- *Traduttore*: può essere visto come l'utente base il quale ha il solo potere di creare delle traduzioni.

Ogni tipologia di utente ha dunque il potere "base" di creare le traduzioni le quali poi appartengono ad un singolo tenant.

2.2.4 API

Di seguito forniamo la lista di API che abbiamo implementato:

- */{tenantId}/translations*
  - *getAllTranslations* : ritorna la lista delle traduzioni appartenenti ad un determinato tenant;
- */{tenantId}/translations /{TranslationKey}*
  - *getTranslation* : ritorna il dettaglio della traduzione;
  - *addTranslation* : aggiunge una traduzione al tenanto specificato;
  - *deleteTranslation* : elimina la traduzione specificata.
- */{tenantId}*
  - *getTenantInfo* : ritorna le informazioni specifiche del tenant;
  - *createTenant* : creazione e/o modifica di un tenant;
  - *deleteTenant* : cancellazione di un tenant;
  - *updateTenant* : modifica di un tenant;
- */{tenantId}/invite*
  - *inviteUser* : invito ai traduttori nel tenant.
- */{tenantId}/user*
  - *createUser* : creazione utente;
  - *deleteUser* : eliminazione utente;
- */tenants*
  - *getAllTenant* : ottenere la lista di tutti i tenant;
- */translations/{language}/{token}*
  - *getTranslationByToken*: ritorna la lista delle traduzioni di un determinato tenant.
- */translations/details/{token}*
  - *getTranslationLibrary*: ritorna la lista delle traduzioni di un determinato tenant e la lingua di default.

2.2.5 Design pattern

Per la gestione di *DynamoDB* e *Cognito* si è scelto di utilizzare il pattern *Singleton*, in quanto permette di avere una sola istanza di un oggetto per tutta la durata dell'applicazione. In questo modo si evita di dover creare più istanze di un oggetto, che potrebbero portare a problemi di sincronizzazione e di memoria. Così facendo si garantisce una sola istanza della classe *DynamoDbHandler* e *CognitoHandler*, le quali sono utilizzate dalle API per interagire con i servizi AWS.

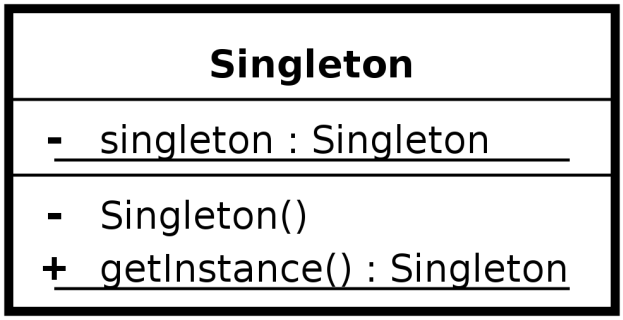

Figura 4: Schema Singleton

2.3 Front-end

2.3.1 Descrizione e pattern architetturale

Per quanto riguarda il Frontend è stato scelto di realizzare la struttura con l'utilizzo del pattern architetturale *Redux*, il quale tramite degli eventi chiamati "azioni" permette di gestire e aggiornare lo stato delle applicazioni. Fornisce inoltre un store centralizzato per gli stati che necessitano di essere utilizzati in tutta l'applicazione, con regole che garantiscano che lo stato venga aggiornato solo in modo prevedibile. Riassumendo, forniamo le principali motivazioni per la suddetta scelta di seguito:

- Il Frontend è realizzato tramite la libreria React, la quale si integra molto bene con il pattern Redux;
- Permette il riutilizzo di vari componenti in diversi contesti senza dover effettuare modifiche;
- Maggiore semplicità per il lavoro in team, ogni singolo componente del gruppo può occuparsi di una sola parte della WebApp;
- Manutenibilità più semplice, dovuta dal disaccoppiamento del codice.

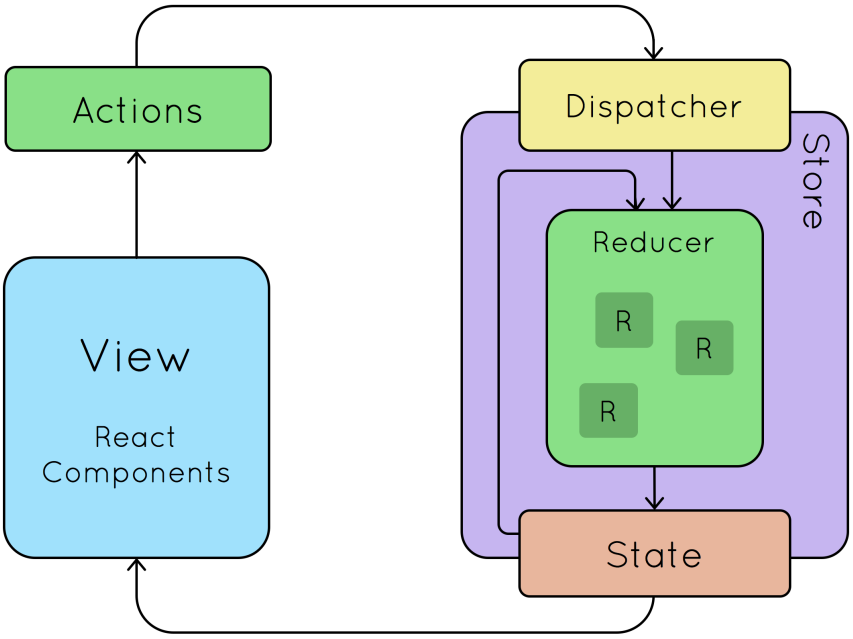

Figura 5: Diagramma - React Redux

Per mantenere separati i diversi ambiti dei componenti, dove possibile questi sono stati divisi in *logic*, che contiene la logica, e *view*, che contiene la parte grafica.

2.3.2 Librerie utilizzate

Per la gestione di alcuni aspetti dell'applicazione frontend sono state utilizzate delle librerie, che riportiamo di seguito:

- *MUI*: libreria che permette di utilizzare componenti già pronti e stilizzati tramite Material UI;
- *react-router-dom*: libreria che permette di gestire le rotte dell'applicazione;
- *Redux Toolkit*: per l'implementazione del pattern *Redux* e la gestione dello stato dell'applicazione;
- *dayjs*: per la formattazione delle date;
- *react-hook-form*: per la gestione dei form.

Inoltre per la gestione delle API abbiamo utilizzato *RTK Query*, uno strumento di data fetching e caching, incluso in *Redux Toolkit* che semplifica le richieste di dati ad un server, memorizza automaticamente i risultati e aggiorna i risultati quando i dati sottostanti cambiano.

2.3.3 Diagrammi delle classi

2.3.3.1 Tipi

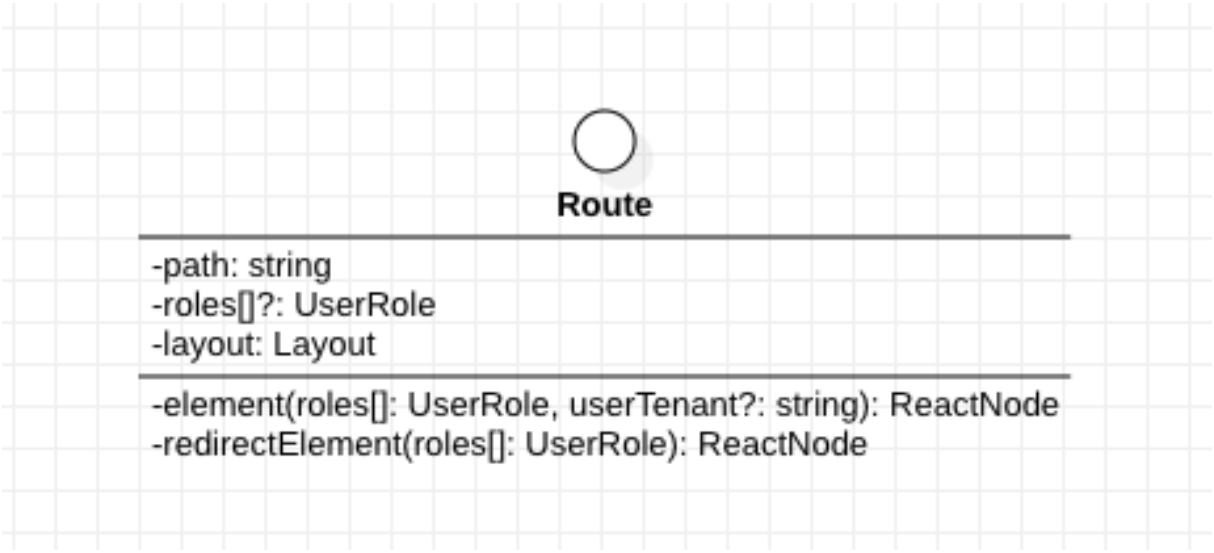

Figura 6: Diagramma - tipo Route

Il tipo *Route* identifica una rotta del sito, ovvero un elemento che indirizza l'utente a una specifica pagina dell'applicazione. E' composta da una path (o percorso) che indica l'indirizzo della pagina, da un insieme di ruoli utente che possono accedere alla pagina e da un layout che indica il layout da utilizzare per la pagina. Le due funzioni *element* e *redirectElement* ritornano rispettivamente la pagina specificata dal percorso e la pagina di default in caso il ruolo dell'utente non sia abilitato ad accedere alla pagina.

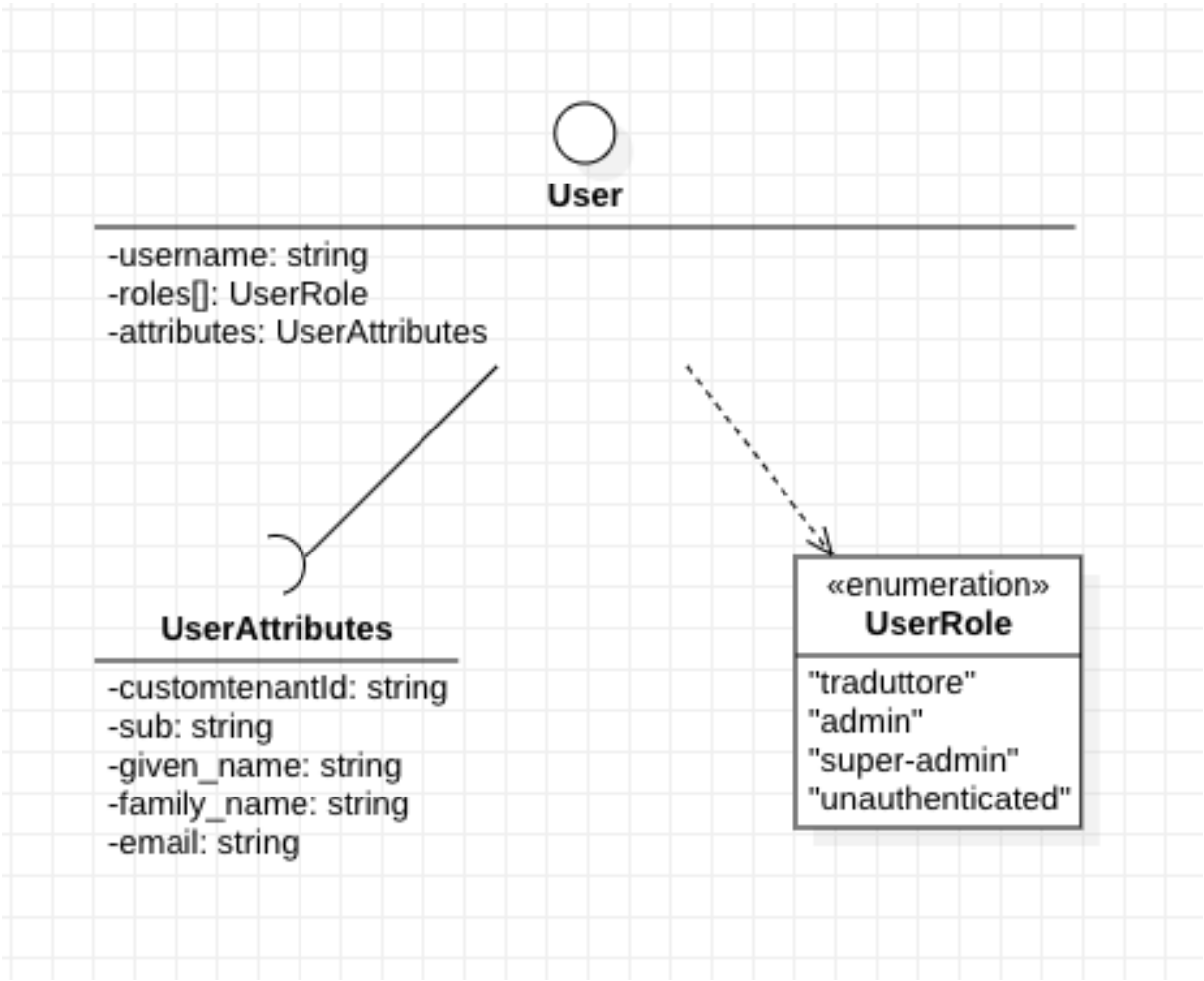

Figura 7: Diagramma - tipi User

Il tipo *User* identifica un utente del sito, caratterizzato da un username, un insieme di ruoli e degli attributi specifici come il tenant di appartenenza, nome e email.

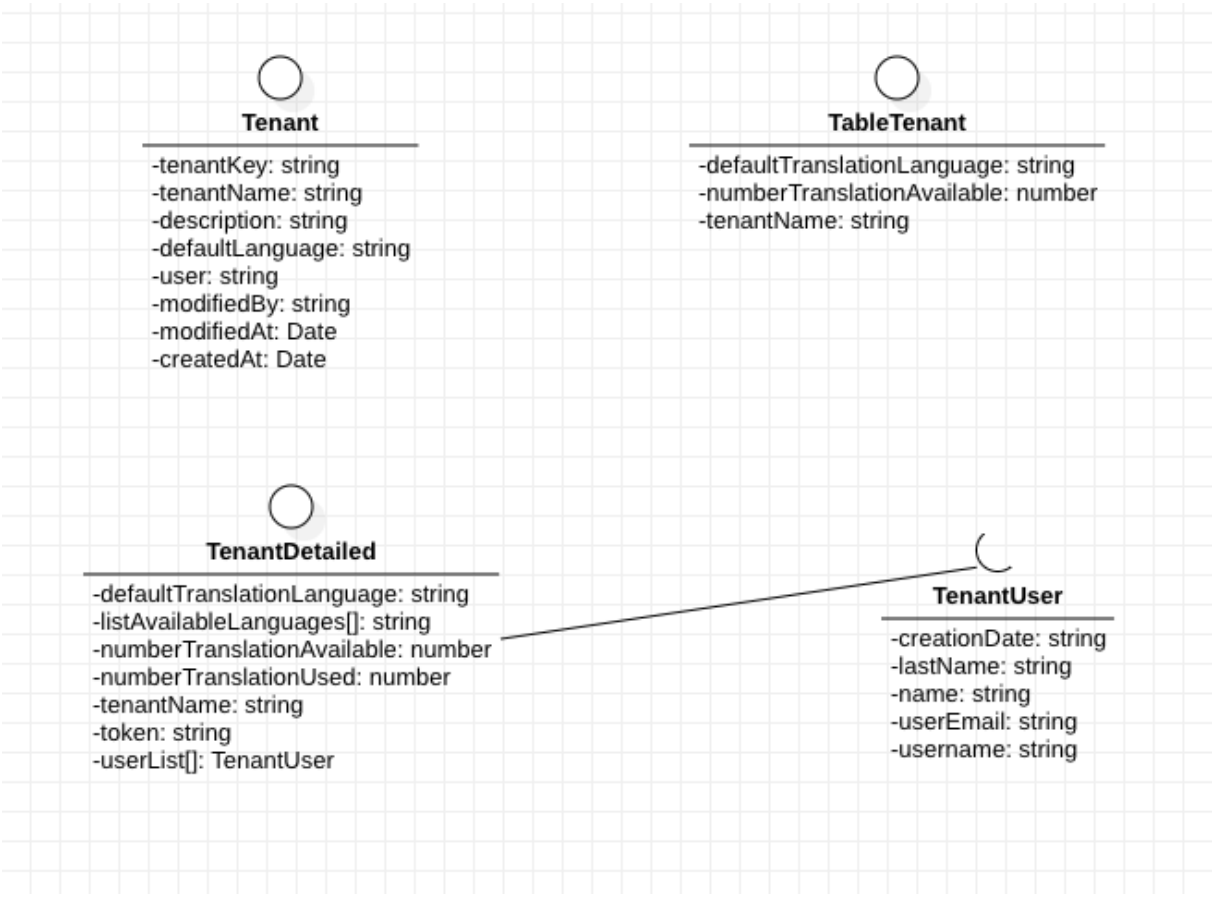

Figura 8: Diagramma - tipi Tenant

I tipi *Tenant* identificano specifici aspetti dei tenant. *Tenant* modella le informazioni generali di un tenant, mentre *TenantDetailed* ne modella le informazioni nel dettaglio. *TableTenant* modella le informazioni di un tenant da mostrare in una tabella, mentre *TenantUser* identifica un utente del tenant.

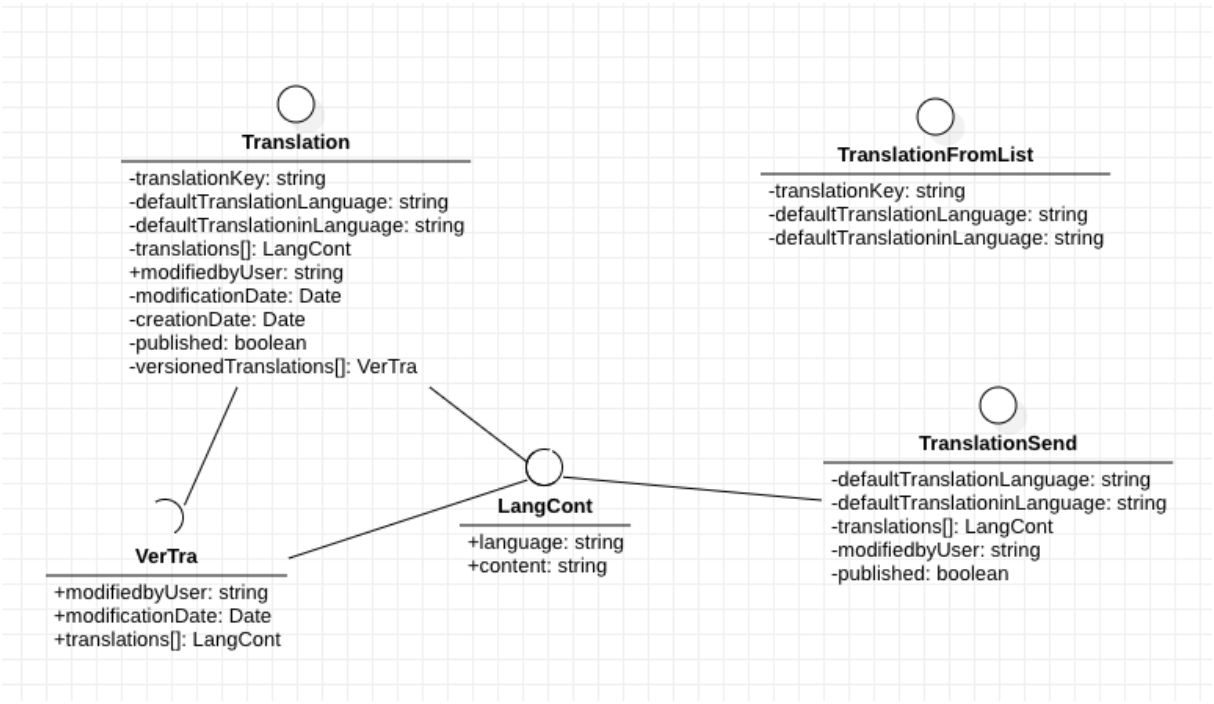

Figura 9: Diagramma - tipi Translation

I tipi *Translation* identificano le traduzioni della piattaforma. *Translation* modella un'intera traduzione, *LangCont* ne modella il contenuto in una specifica lingua, mentre *VerTra* ne modella una versione precedente.
TranslationFromList modella le informazioni di una traduzione da mostrare in una tabella, e *TranslationSend* identifica una traduzione da inviare al server.

2.3.3.2 Hooks

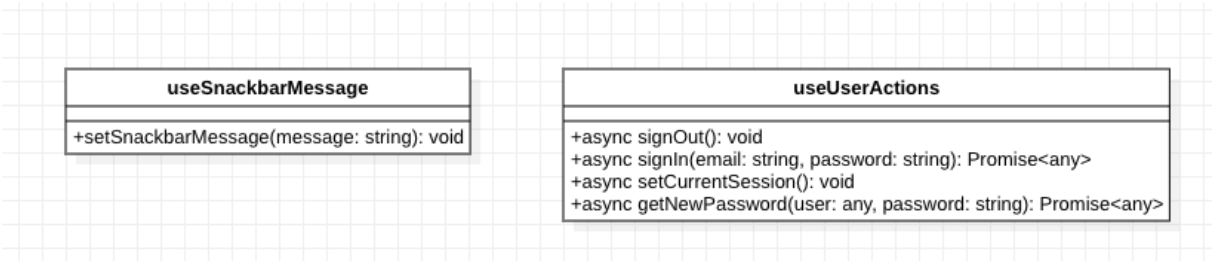

Figura 10: Diagramma - Hooks

L'applicazione utilizza due hooks per gestire le azioni e lo stato dell'applicazione. *useUserActions* raccoglie e rende disponibili una serie di funzioni di AWS Amplify per gestire le azioni degli user. *useSnackbarMessage* consente l'apertura di una snackbar per mostrare messaggi di errore all'utente.

2.3.3.3 App

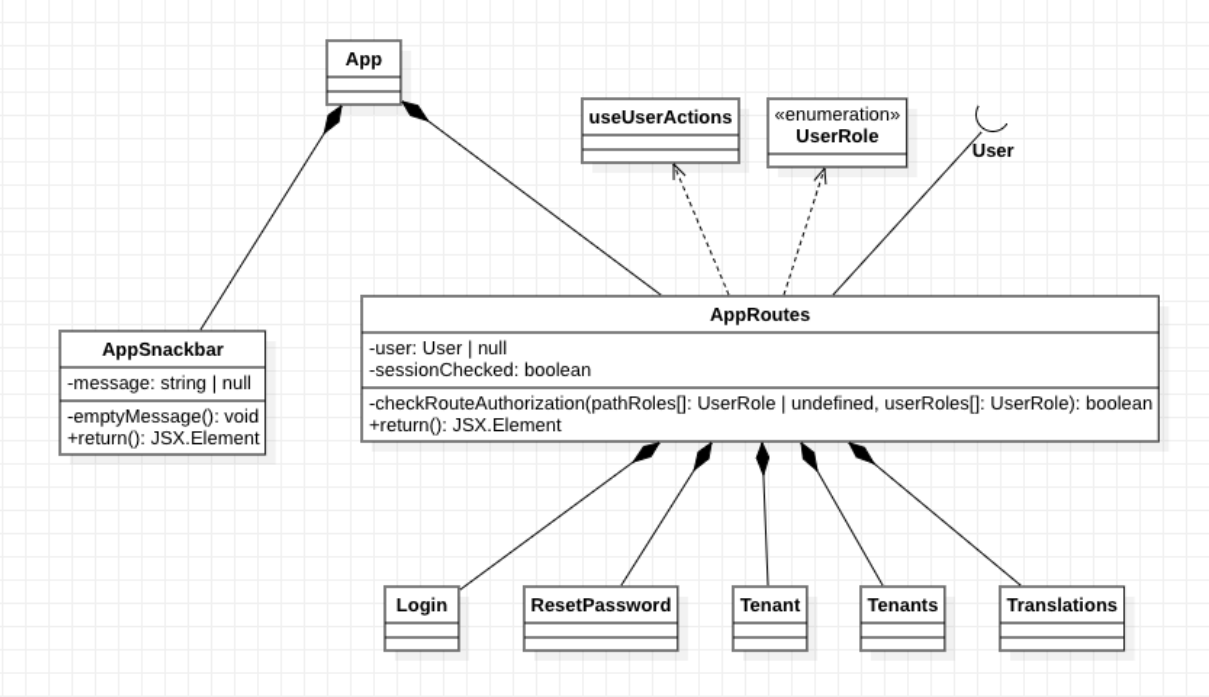

Figura 11: Diagramma - App

Questo schema identifica l'app, composta da due viste:

- *AppSnackbar*, che gestisce la snackbar per mostrare messaggi di errore all'utente;
- *AppRoutes*, che gestisce le rotte dell'applicazione, che possono visualizzare una di 5 pagine principali:
  - *Login*, che gestisce il login dell'utente;
  - *ResetPassword*, che gestisce il reset della password;
  - *Tenants*, che gestisce la visualizzazione dei tenant;
  - *Tenant*, che gestisce la visualizzazione di un tenant;
  - *Translations*, che gestisce la visualizzazione delle traduzioni;

2.3.3.4 Login

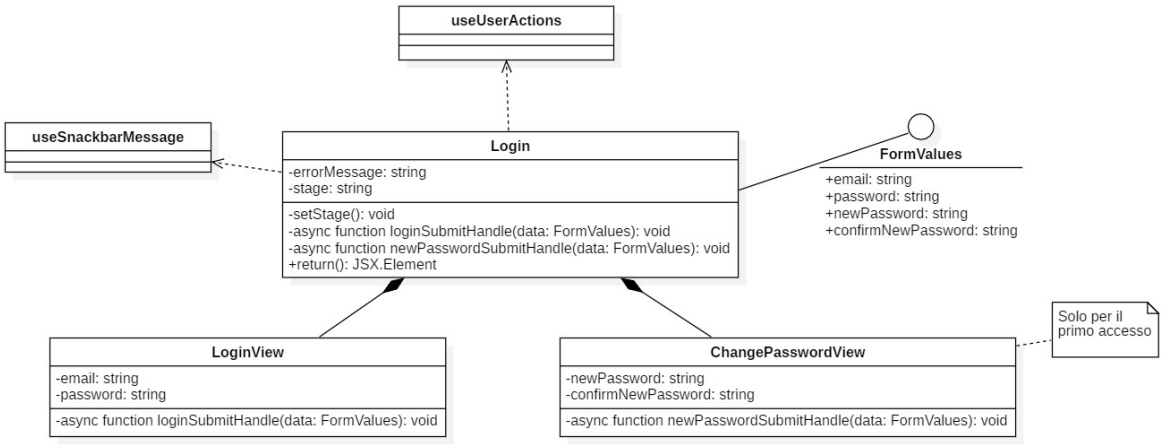

Figura 12: Diagramma - componente Login

La pagina *Login* è caratterizzata da due viste differenti:

- *LoginView*, corrispondente al form di accesso all'applicativo;
- *ChangePasswordView*, corrispondente alla funzionalità "Password dimenticata?".

Questa pagina utilizza entrambi gli hooks *useUserActions* e *useSnackbarMessage*.

2.3.3.5 Reset Password

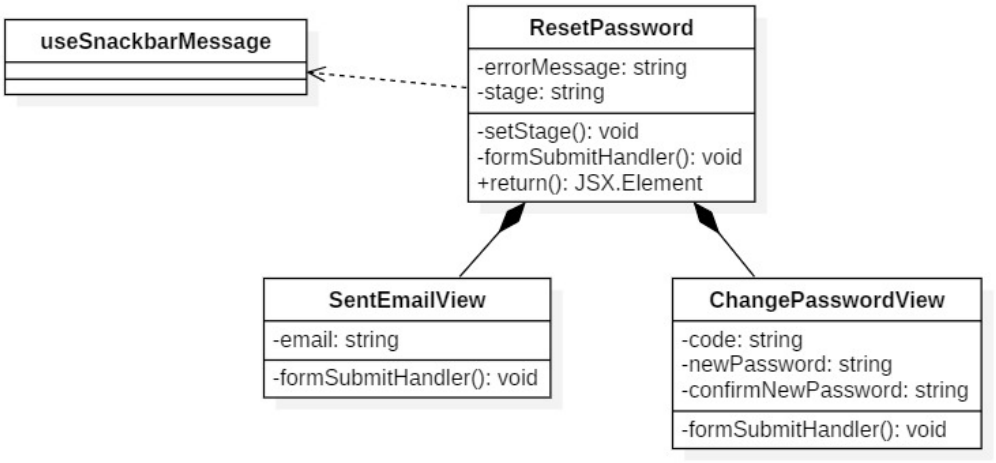

Figura 13: Diagramma - componente ResetPassword

La pagina *ResetPassword* è caratterizzata da due viste distinte che corrispondono ai due step attraversati per il cambio della password:

- *SentEmailView*, corrisponde ad un form dove viene chiesto di inserire la mail legata all'account di cui si desidera recuperare la passsword;

- *ChangePasswordView*, corrisponde ad un form dove viene chiesto di inserire il codice di recupero (invitato alla mail inserito nello step precedente), la nuova password e la conferma della nuova password.

Questa pagina utilizza l'hook *useSnackbarMessage*.

2.3.3.6 Tenants

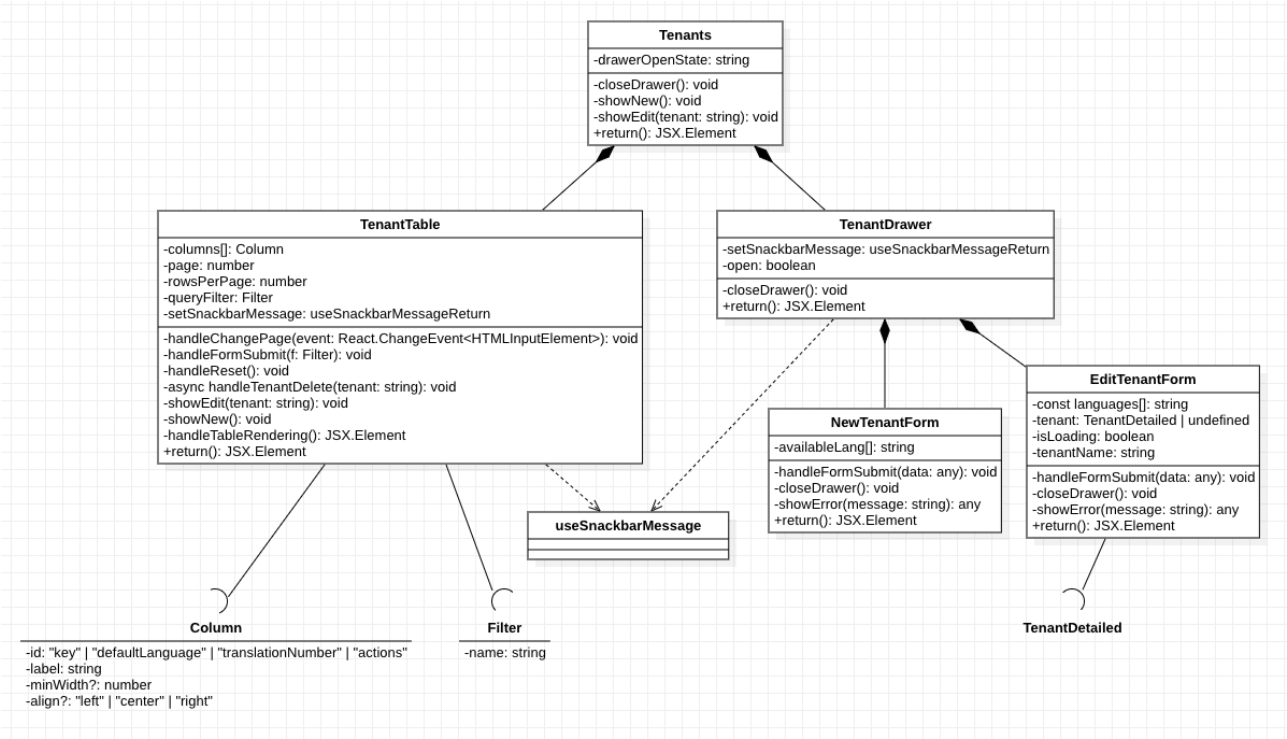

Figura 14: Diagramma - componente Tenants

La pagina *Tenants* è caratterizzata da due viste principali:

- *TenantTable*, che visualizza una lista dei tenant presenti all'interno dell'applicazione, e permette di aggiungerne di nuovi;
- *TenantDrawer*, un drawer che si apre dall'alto per visualizzare uno di due form:
  - *NewTenantForm*, che permette di aggiungere un nuovo tenant;
  - *EditTenantForm*, che permette di modificare un tenant esistente.

Entrambe le viste utilizzano l'hook *useSnackbarMessage*.

2.3.3.7 Tenant

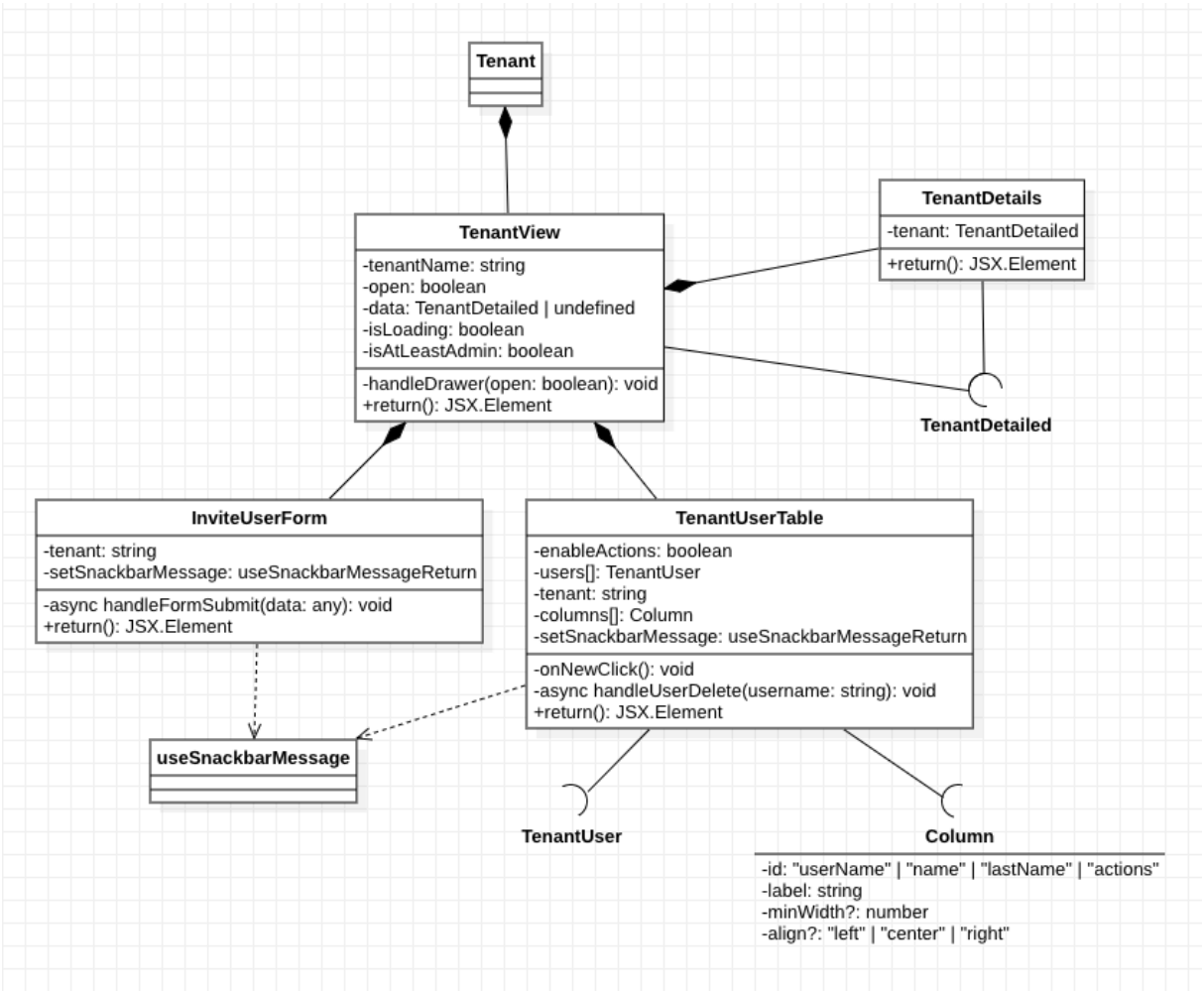

Figura 15: Diagramma - componente Tenant

La pagina *Tenant* è caratterizzata da una vista, *TenantView*, che visualizza le informazioni in dettaglio di un tenant e i suoi utenti, tramite tre componenti:

- *TenantDetails*, che visualizza le informazioni dettagliate del tenant;
- *TenantUserTable*, che visualizza una lista degli utenti del tenant;
- *InviteUserForm*, un drawer che si apre dall'alto con un form per invitare un nuovo utente al tenant.

I due componenti *InviteUserForm* e *TenantUserTable* utilizzano l'hook *useSnackbarMessage*.

2.3.3.8 Translations

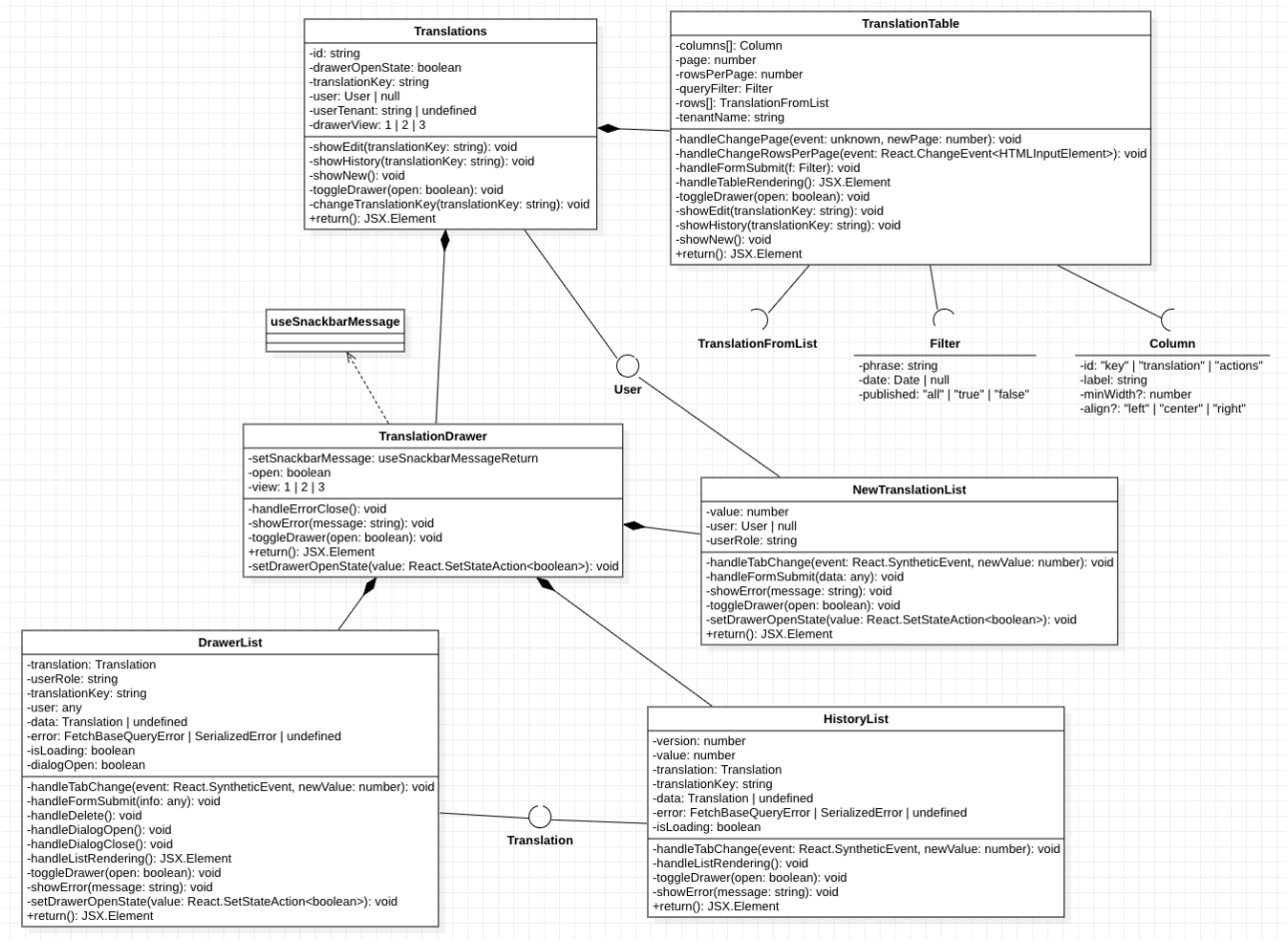

Figura 16: Diagramma - componente Translations

La pagina *Translations* permette la visualizzazione delle traduzioni di un tenant, e anche l'esecuzione di alcune operazioni su di esse, come visualizzazione, modifica ed eliminazione. La pagina si compone di due viste principali:

- *TranslationsTable*, che visualizza una lista delle traduzioni del tenant;
- *TranslationDrawer*, un drawer che si apre dall'alto per visualizzare uno di tre componenti:
  - *NewTranslationList*, che contiene un form per aggiungere una nuova traduzione al tenant;
  - *DrawerList*, che permette la modifica di una traduzione;
  - *HistoryList*, che permette la visualizzazione delle cinque ultime versioni di una traduzione;

La vista *TranslationDrawer* usa l'hook *useSnackbarMessage*.

2.4 Libreria

La libreria che abbiamo creato è un wrapper che permette di recuperare i dati delle traduzioni dal backend. Tramite un token identificativo, è possibile effettuare chiamate per ottenere traduzioni pubblicate in una specifica lingua, oltre che informazioni riguardanti il tenant.

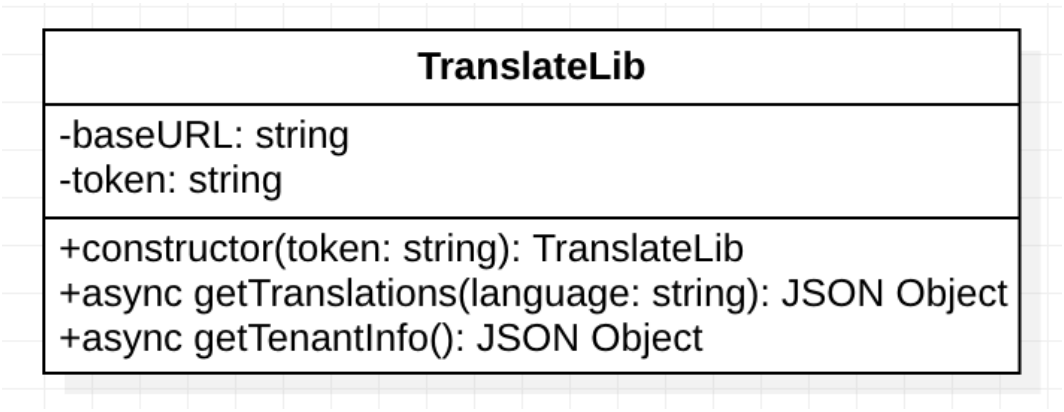

Figura 17: Diagramma - Libreria

3 Requisiti soddisfatti

3.1 Tabella dei requisiti soddisfatti

Tabella 1: Tabella dei Requisiti soddisfatti

Fonte	Requisiti	Fonte	Requisiti
R1FI1	soddisfatto	R1FC2	soddisfatto
R1FI2	soddisfatto	R1FC3	soddisfatto
R1FI3	soddisfatto	R1FC4	soddisfatto
R1FI4	soddisfatto	R1FC5	soddisfatto
R1FI5	soddisfatto	R1FC6	soddisfatto
R1FI6	soddisfatto	R2FI17	soddisfatto
R1FI7	soddisfatto	R2FI18	non soddisfatto
R1FI8	soddisfatto	R2FI19	soddisfatto
R2FI9	soddisfatto	R2FI20	soddisfatto
R2FI10	soddisfatto	R2FI21	soddisfatto
R2FI11	soddisfatto	R1FZ9	soddisfatto
R2FI12	soddisfatto	R2FI22	soddisfatto
R1FZ1	soddisfatto	R2FI23	soddisfatto
R1FZ2	soddisfatto	R2FI24	soddisfatto
R1FZ3	soddisfatto	R2FI25	soddisfatto
R1FZ4	soddisfatto	R2FI26	soddisfatto
R1FZ5	soddisfatto	R2FI27	non soddisfatto
R1FZ6	soddisfatto	R2FI28	soddisfatto
R1FZ7	soddisfatto	R2FI29	non soddisfatto
R1FI13	soddisfatto	R1FI30	soddisfatto
R1FI14	soddisfatto	R1FI31	soddisfatto
R2FI15	soddisfatto	R2FI32	soddisfatto
R1FI16	soddisfatto	R1FI33	soddisfatto
R1FZ8	soddisfatto	R1FI34	soddisfatto
R1FC1	soddisfatto	R1FI35	soddisfatto
R1FI36	soddisfatto	R1FI46	soddisfatto
R1FZ10	soddisfatto	R1FI47	soddisfatto
R1FI37	soddisfatto	R1FI48	soddisfatto

Fonte	Requisiti	Fonte	Requisiti
R1FI38	soddisfatto	R1FI49	soddisfatto
R1FI39	soddisfatto	R1FI50	soddisfatto
R1FI40	soddisfatto	R1FI51	soddisfatto
R1FI41	soddisfatto	R1FI52	soddisfatto
R1FI42	soddisfatto	R1FI53	soddisfatto
R1FI43	soddisfatto	R1FI54	soddisfatto
R1FI44	soddisfatto	R1FI55	soddisfatto
R1FC7	soddisfatto	R1FI56	soddisfatto
R1FI45	soddisfatto		

3.2 Grafici dei requisi soddisfatti

3.2.1 Requisiti soddisfatti vs non soddisfatti

Il seguente grafico a torta rappresenta le percentuali di requisiti soddisfatti contro quelli che non lo sono. Nello specifico 70 di questi sono stati soddisfatti e i rimanenti 3 invece no.

Figura 18: Diagramma - Requisiti soddisfatti vs non soddisfatti

3.2.2 Requisiti obbligatori soddisfatti vs non soddisfatti

Di seguito viene riportato il grafico a torta rappresentante i requisiti obbligatori che il team è riuscito a soddisfare.

Figura 19: Diagramma - Requisiti obbligatori soddisfatti

Come si può notare dal grafico, il team è riuscito a soddisfare tutti e 55 i requisiti obbligatori.