# DOCUMENTO 4 DESIGN

## 4.1 INTRODUZIONE

In questo documento andremo a descrivere come abbiamo strutturato il nostro progetto, andando a definire un'architettura software e successivamente definendo il Design in dettaglio. L'obiettivo è quella di dare una visione completa e dettagliata sulla struttura statica e il comportamento dinamico del nostro programma.
I diagrammi uml avranno un approccio top-down: dalle funzionalità utente (Use Case) all'architettura logica (Package e Component), fino ai dettagli delle classi e delle interazioni (Class, Sequence, State Machine).

## 4.2 Software Architecture

Il sistema adotta un'Architettura a Strati, l'abbiamo scelta per garantire una separazione delle responsabilità rispetto alle varie funzionalità che il programma doveva svolgere e per facilitare la manutenibilità del codice. Inoltre, data la natura del programma, l'architettura integra il pattern Event-Driven (reattivo) per la sincronizzazione dei dati: l'interfaccia utente non richiede attivamente i nuovi messaggi , ma reagisce automaticamente agli eventi generati dal database remoto.

L'architettura si articola in tre layer principali:

1. Presentation Layer (UI): Gestisce l'interazione con l'utente e la visualizzazione.
2. Service Layer ( Logica): Contiene tutta la logica del progetto, ogni qualvolta che viene effettuata una azione la gestisce.
3. Data Layer (infrastruttura): Gestisce la persistenza e le comunicazioni esterne, in questo livello ci sono tutte le comunicazioni con firebase, con il FileSystem e con le API ARASAAC.

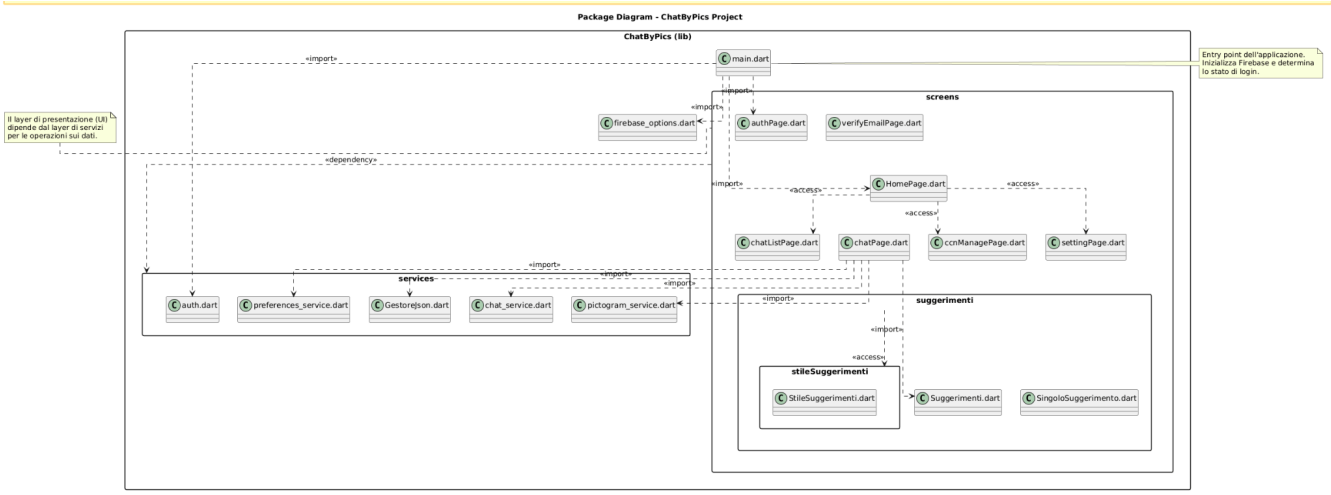
### 4.2.1 Organizzazione dei Package(Vista Logica)

Il codice sorgente è organizzato modularmente. Ogni layer comunica solo con quello sottostante, rispettando il principio di inversione delle dipendenze.

1. screens (UI Layer):
   -Contiene i widget Flutter che compongono le schermate
   -Responsabilità: Catturare l'input utente come click e inserimento testo e renderizzare lo stato fornito dai servizi. Non contiene logica di accesso diretto al database.
2. services (Service Layer):
   -Contiene le classi che implementano la logica e fungono da ponte verso il backend.
   -Responsabilità: Fornire metodi puliti alla UI, nascondendo la complessità delle chiamate API o delle query al database.

3. models (Data Layer): gestisce effettivamente la parte di comunicazione con il database

# PACKAGE DIAGRAM




(visualizzabile nella cartella con tutti i diagrammi)

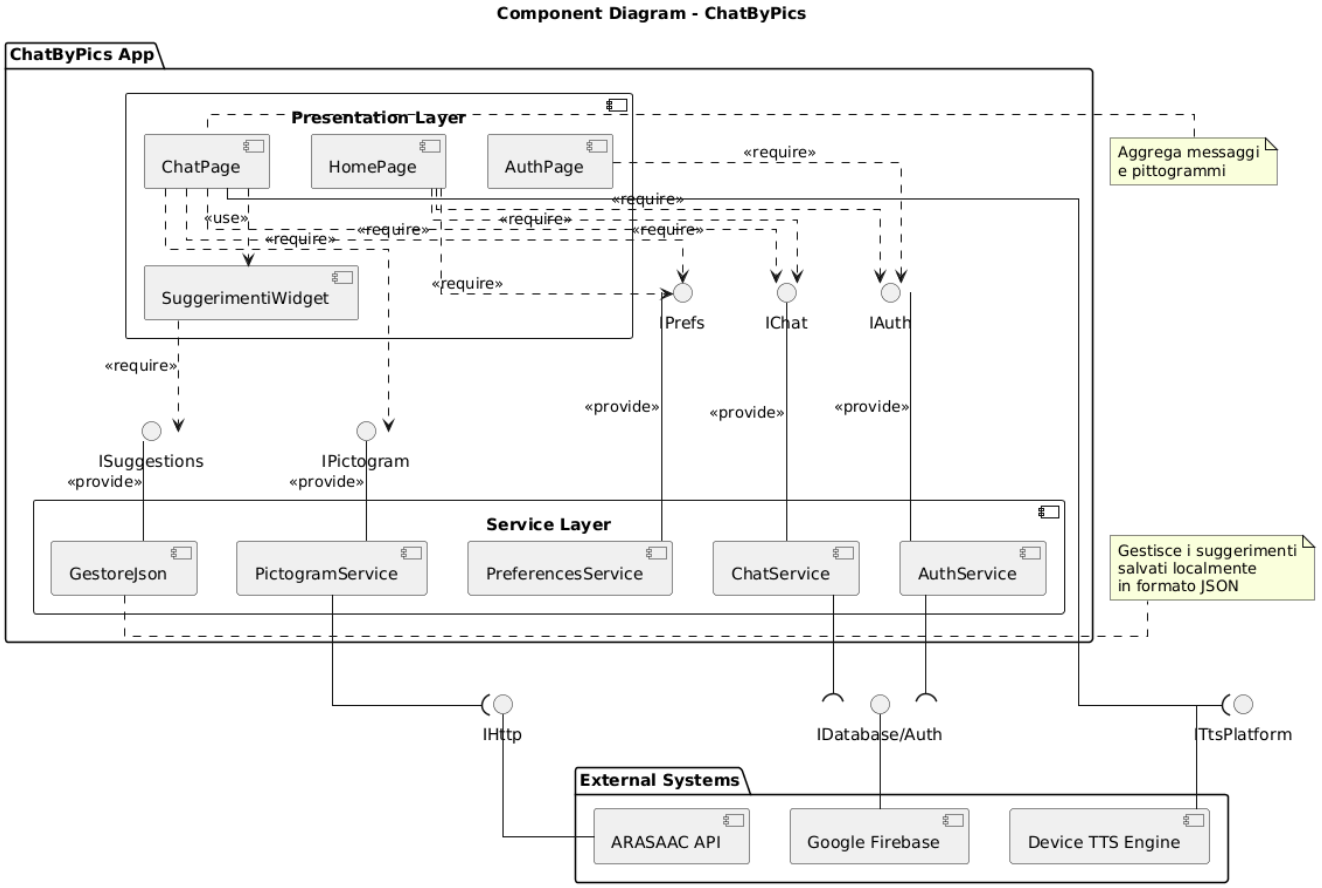
# 4.2.2 Componenti (Vista Fisica)

Il sistema è distribuito e si basa sull'interazione tra il client mobile e servizi cloud eterogenei.

1. Mobile Client : Il componente principale eseguito sul dispositivo dell'utente. Gestisce la logica locale, la cache e l'interfaccia.
2. Authentication Provider : Gestisce l'identità degli utenti (Tutor e Utenti standard), i token di sessione e il recupero password sicuro, in questo caso Firebase Auth
3. Real-time Database : Componente critico per la persistenza.
   -Memorizza profili utenti, chat e messaggi.
   -Utilizza il protocollo WebSocket per inviare aggiornamenti push al client in tempo reale, in questo caso Cloud Firestore.
4. External API Provider (Arasaac): Servizio di terze parti interrogato via HTTP REST.

   -Fornisce le risorse grafiche (pittogrammi) e i metadati per la ricerca semantica (es. categorie, traduzioni).

COMPONENT DIAGRAM



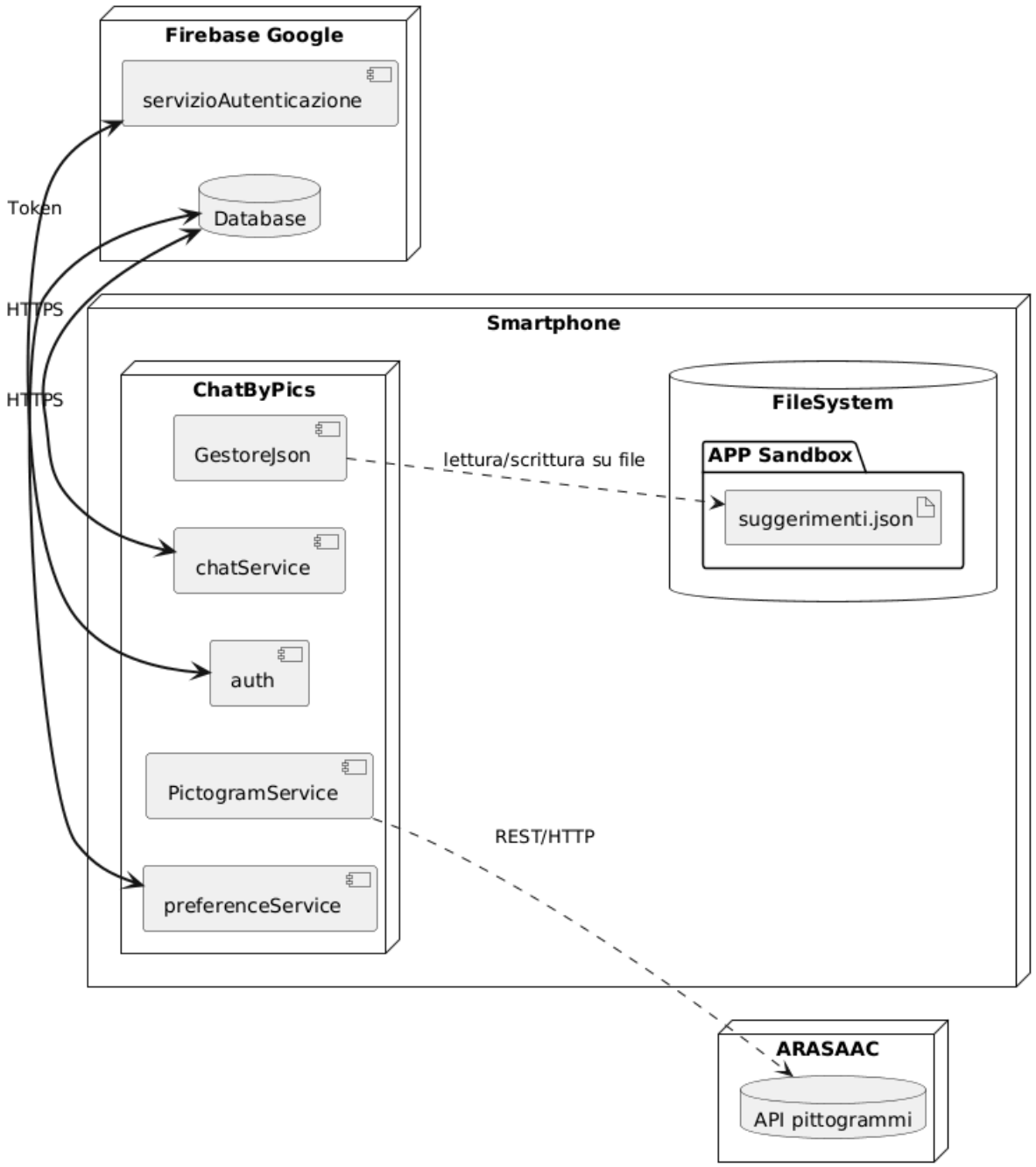

# 4.2.3 Distribuzione (Vista Deployment)

Vista della distribuzione della applicazione, l'infrastruttura e distribuita su 3 nodi:

Smartphone, il client dove è in esecuzione l'applicativo, nel suo fileSystem nelle cartelle private dell'applicazione sarà presente il file suggerimenti.json contenente i dati dei suggerimenti dei pittogrammi.

Servizio firebase Google, costituisce il back-end dell'applicazione, ospitando il servizio di autenticazione e repository dei messaggi e dei profili utente.

Servizio API ARASAAC, fornisce le API per il recupero dei pittogrammi.




## 4.2.4 GESTIONE DIPENDENZE E LIBRERIE

Avendo fatto un’applicazione in flutter non abbiamo potuto usare delle librerie con Maven, abbiamo comunque usato il suo equivalente le principali librerie esterne integrate nel pubspec.yaml sono:

1. firebase_core & cloud_firestore: Driver ufficiali per la connessione al Backend-as-a-Service (BaaS) di Google. Garantiscono la persistenza dei dati e la sincronizzazione offline-online.
2.http: Client HTTP leggero utilizzato da PictogramService per effettuare chiamate GET asincrone verso l'API pubblica di Arasaac.

3. firebase_auth: utilizzato per la gestione del login e sistema token
4. path_provider: utilizzato per l’accesso al FileSystem per accedere alle cartelle private dell’applicazione(Sandbox) indipendentemente dalla piattaforma Ios/Android
5.flutter_tts: utilizzato per la sintesi vocale

# 4.3 DESIGN

Tutti i diagrammi utilizzati per il design applicativo sono posizionati in una cartella specifica, Codice PlantUML e Immagini. (CARTELLA: DiagrammiUML)

## 4.3.1 MODELLO STATICO

#### 4.3.1.1 CLASS DIAGRAM

Il Class Diagram rappresenta la struttura statica del sistema, evidenziando le classi principali, i loro metodi e le relazioni di dipendenza verso i servizi esterni. In coerenza con l'architettura a strati definita in precedenza, il diagramma si focalizza sulle classi del UI layer, e fanno vedere come queste si interfacciano con il service layer

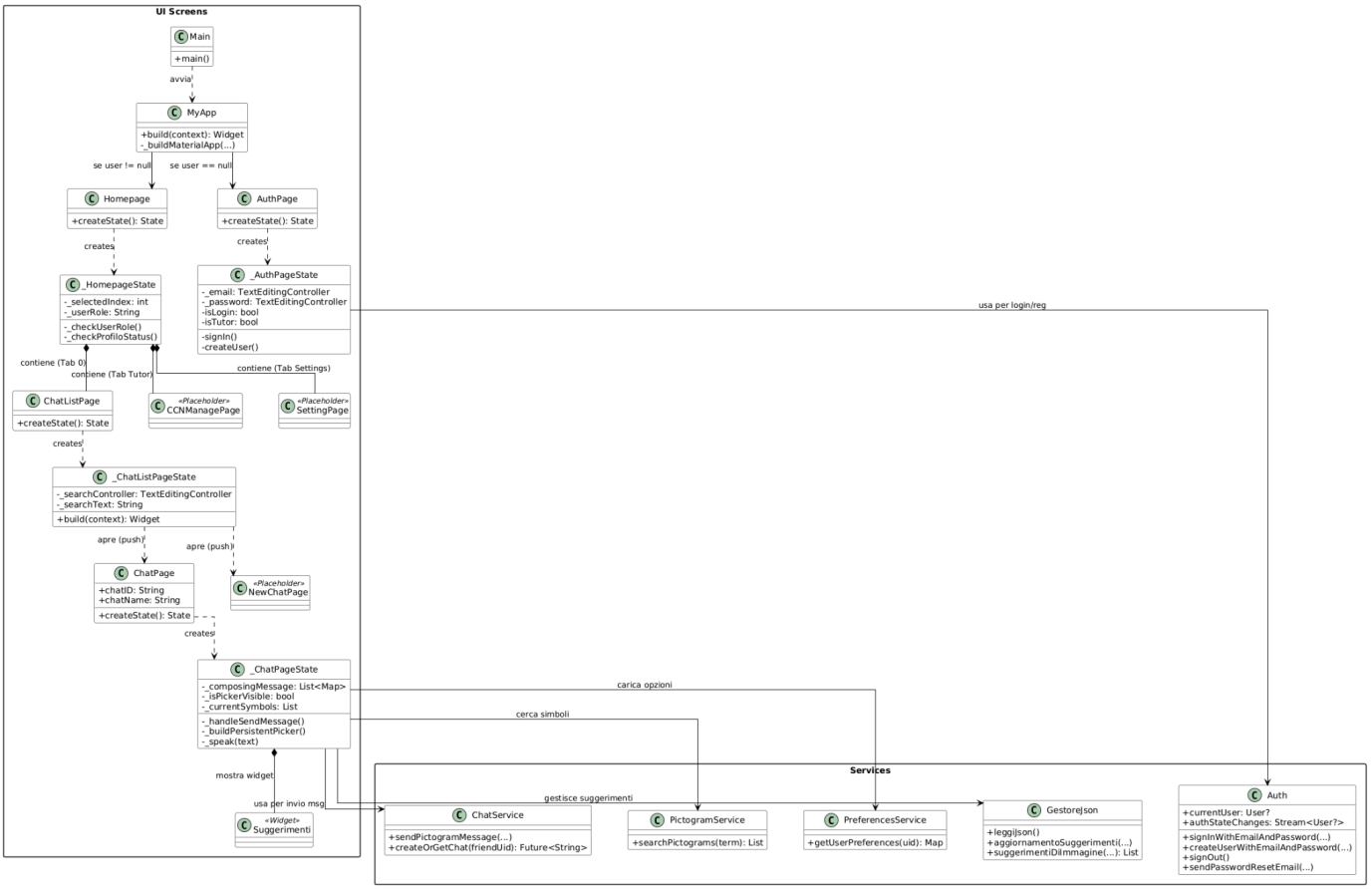
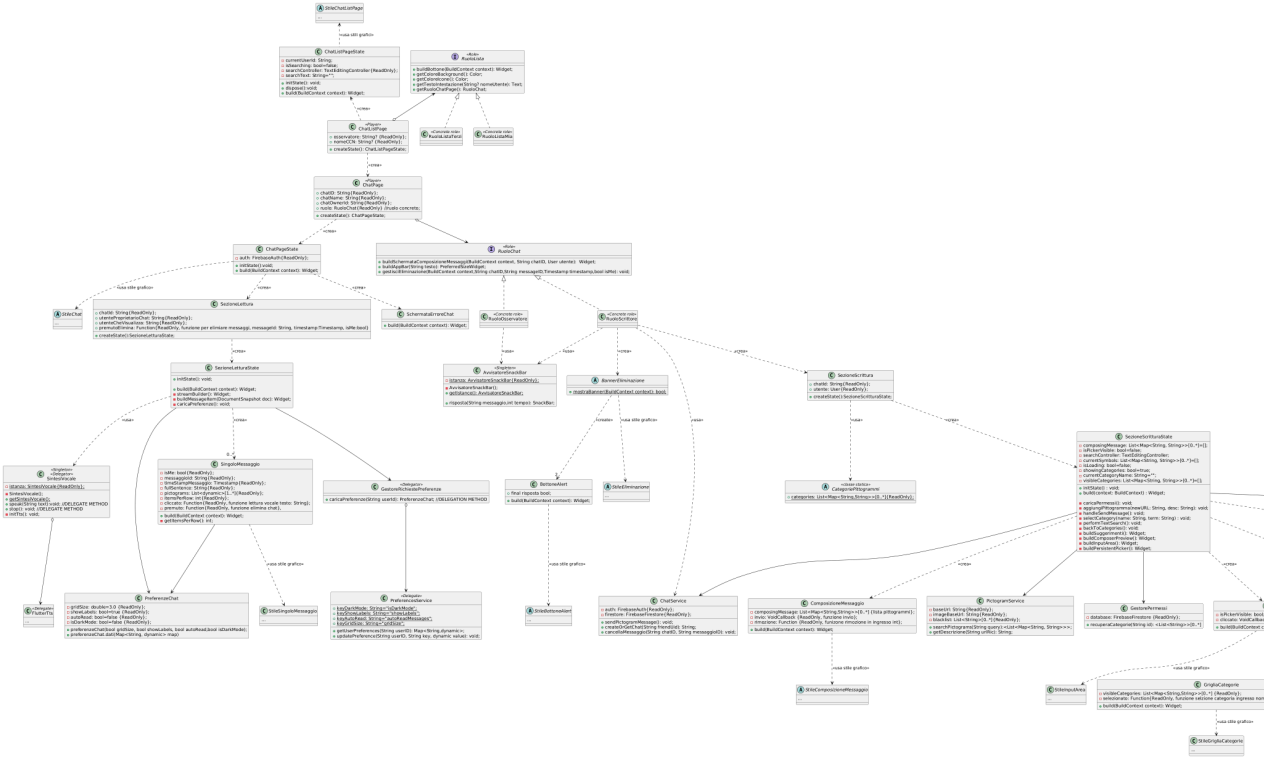



**DIAGRAMMA CLASSI CHAT PAGE E CLASSI CORRELATE**

Il diagramma rappresenta la struttura delle classi per la chatPage e le classi che interagiscono con questa. Per un'osservazione più dettagliata invitiamo il lettore ad eseguire il codice PlantUML che si trova nella cartella dedicata ai diagrammi.



# 4.3.2 MODELLO DINAMICO

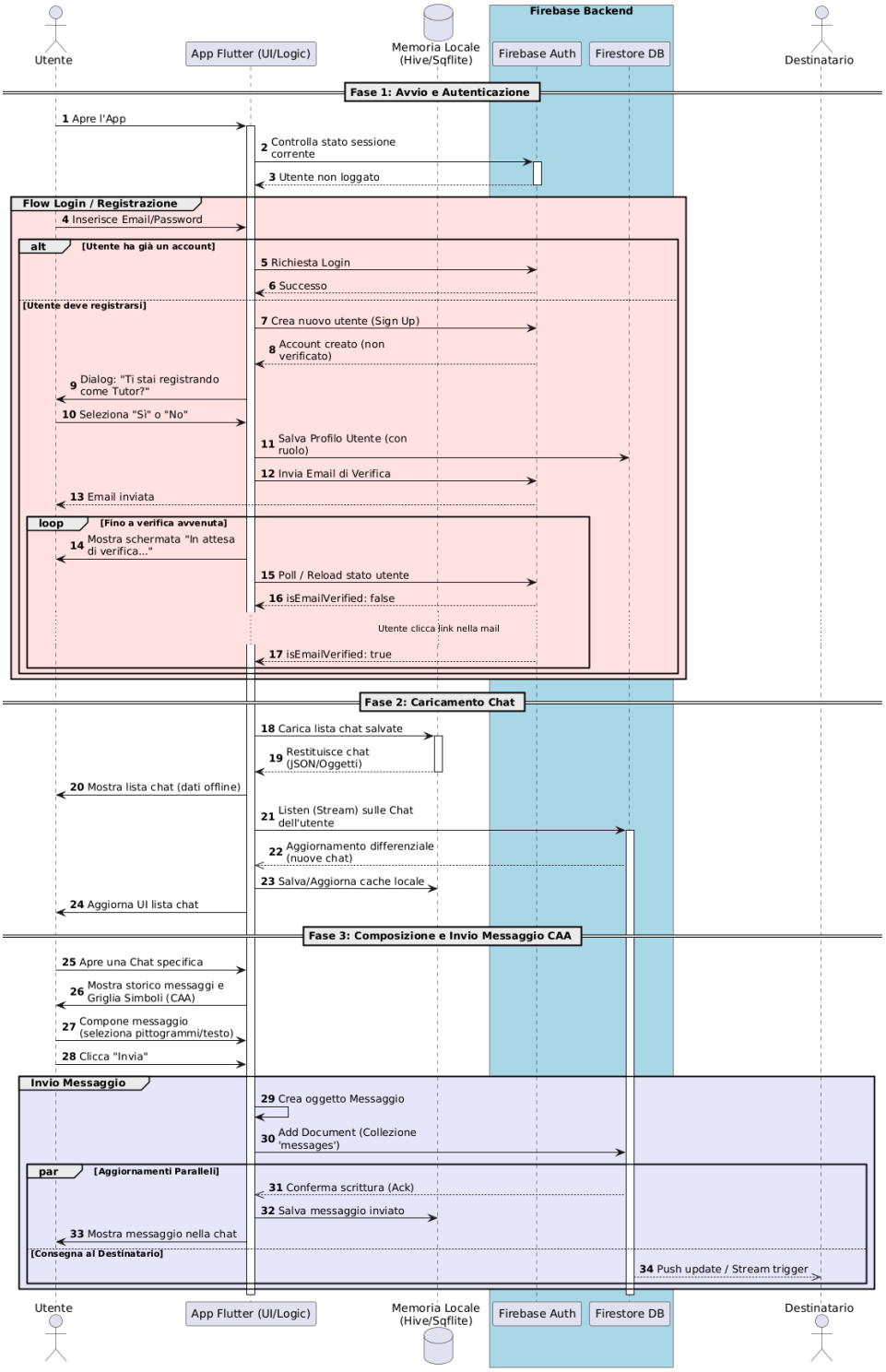
### 4.3.2.1 SEQUENCE DIAGRAM

Visione Generale

Per descrivere le varie interazioni che compiono gli oggetti del nostro sistema useremo 2 uml, il primo useremo un sequence diagram, in cui vedremo la sequenza di chiamate e di messaggi da quando accede a quando invia un messaggio, fino a quando il destinatario lo riceve

Riporto la parte successiva all'autenticazione con invio del messaggio da parte di un utente

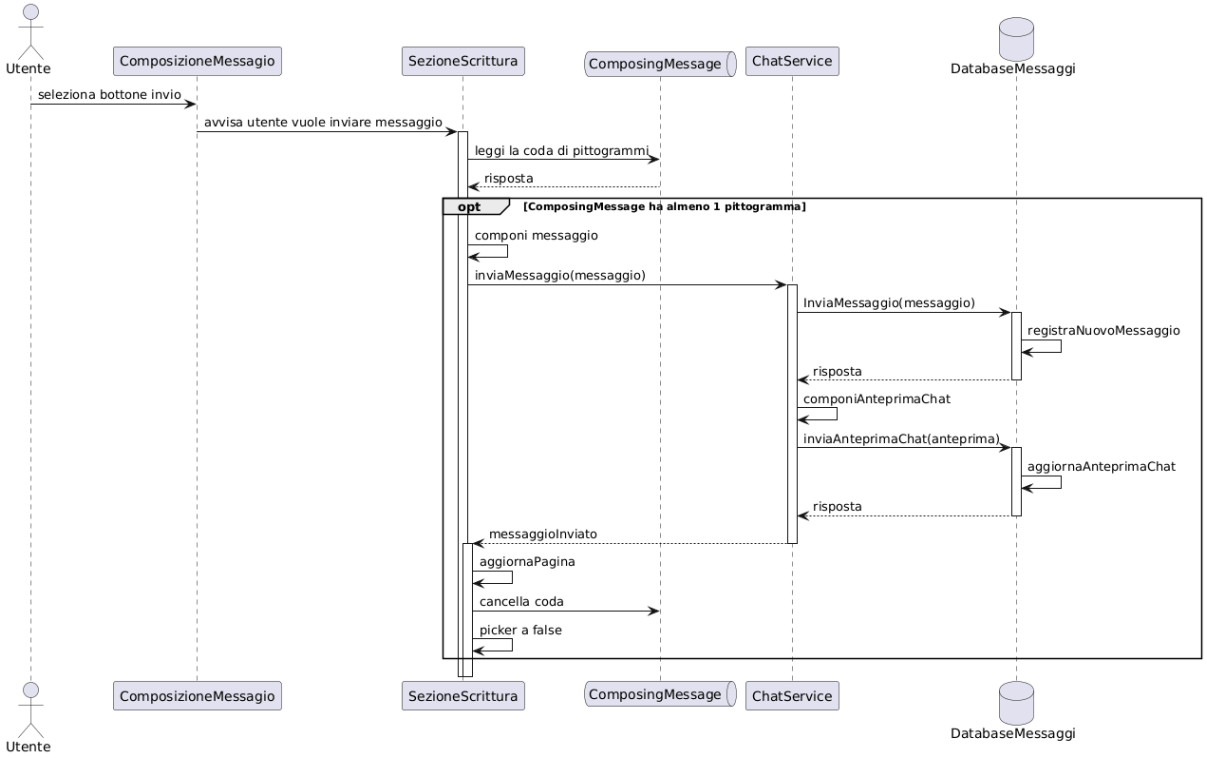
1. L'utente preme "Invia" nella ChatPage.
2. La ChatPage chiama il metodo sendPictogramMessage di ChatService.
3. ChatService costruisce il payload JSON contenente la lista dei pittogrammi e i metadati.
4. Il servizio invia i dati alla collezione messages su Firestore.
5. Parallelamente, aggiorna il documento della chat (campi lastMessageTime e lastMessageData) per l'anteprima nella Home.
6. Firestore notifica i client in ascolto del nuovo dato inserito.




## Gestione invio Messaggio

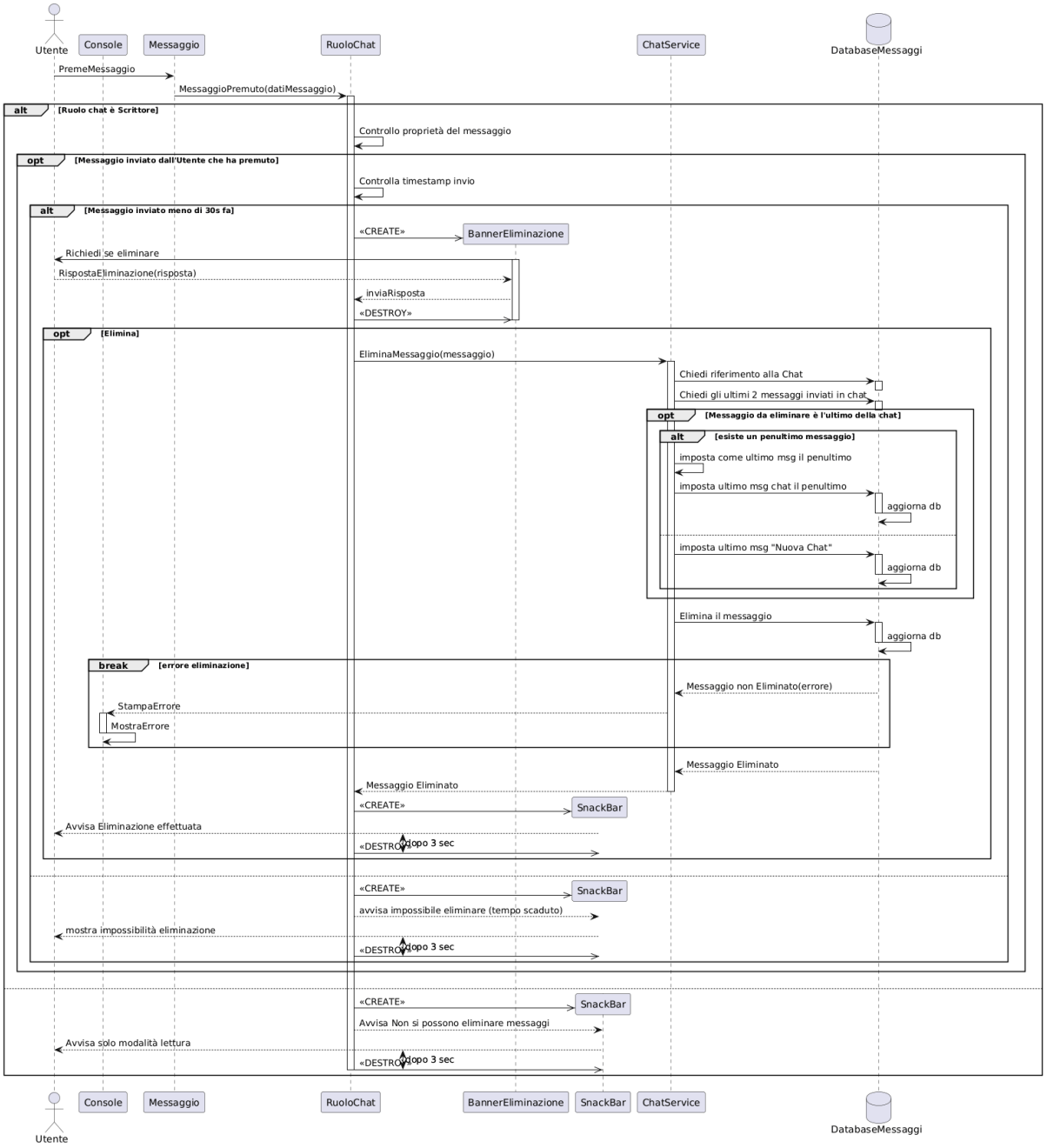
Interazioni tra i vari componenti nell'invio di un messaggio da parte di un utente. mette in evidenza che la parte ComposizioneMessaggio non possiede logica ma è sono UI/Grafica che mostra la sezione dedicata alla composizione del messaggio e che avvisa la parte logica dedicata alla gestione delle funzionalità per la scrittura dei messaggi degli eventi che succedono. La sezioneScrittura andrà a gestire gli eventi, in questo caso  utilizzando la

classe ChatService(Back-end) per l’interazione con il Database. Con ComposingMessage intendiamo la parte dedicata all’immagazzinamento dei pittogrammi da inviare che è una lista di pittogrammi.




## Eliminazione Messaggio

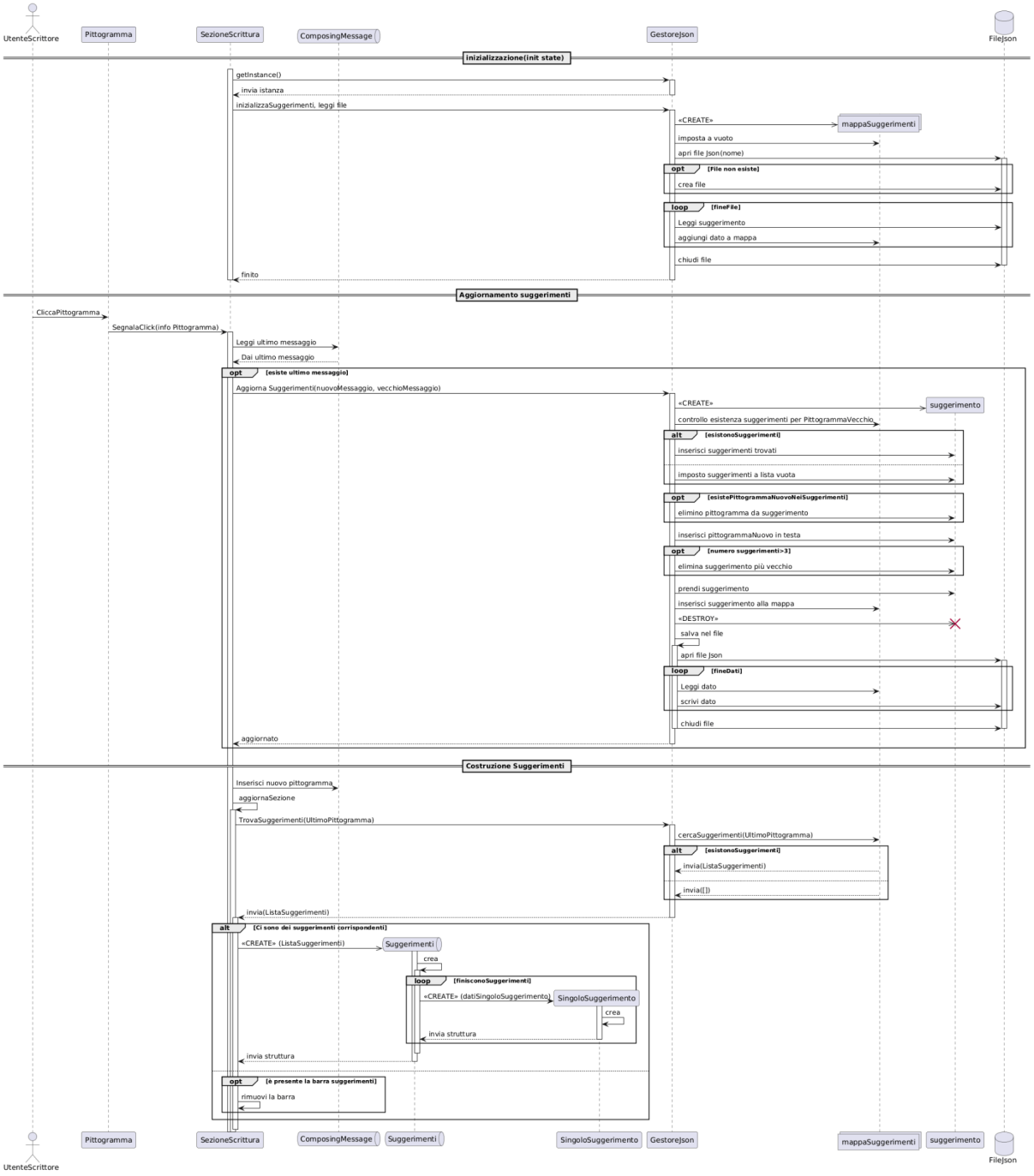
Mostra le interazioni tra i componenti nel caso in cui un utente decida di eliminare un messaggio(tramite la pressione dello stesso). Viene mostrato come la gestione logica della eliminazione del messaggio sia svolta dal ruolo della chat, questo perchè in base a che tipologia di Ruolo assunto dalla chat (Osservatore o Scrittore) si avranno 2 comportamenti differenti per l’eliminazione.




## Gestione Suggerimenti

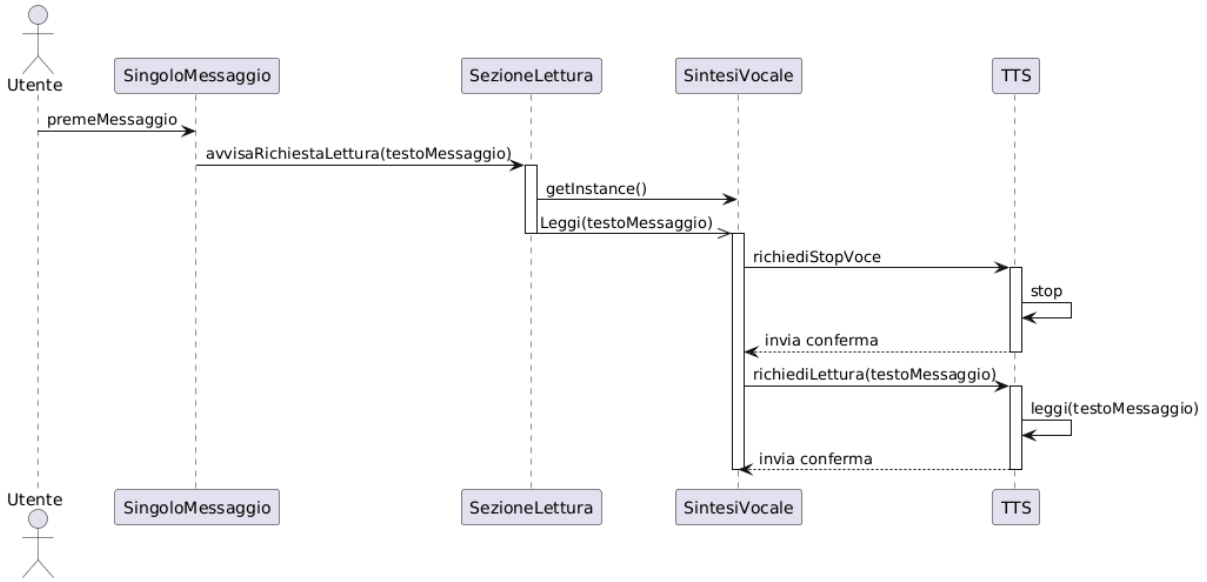
Il diagramma mostra le interazioni tra i vari componenti per gestire i suggerimenti dei pittogrammi forniti all'utente quando inserisce i pittogrammi nella chat.
Sono presenti 3 fasi per la gestione dei suggerimenti:

1. inizializzazione: viene creata la mappa dei suggerimenti in memoria leggendo dal file json i suggerimenti salvati.
2. aggiornamento suggerimenti: nel caso in cui un utente selezioni un pittogramma la SezioneScrittura(parte logica) andrà, utilizzando il gestoreJson, ad aggiornare la mappa dei suggerimenti e il file con i suggerimenti.
3. costruzione suggerimenti: dopo aver inserito un pittogramma sarà necessario mostrare i suggerimenti per quel pittogramma.




## Sintesi Vocale

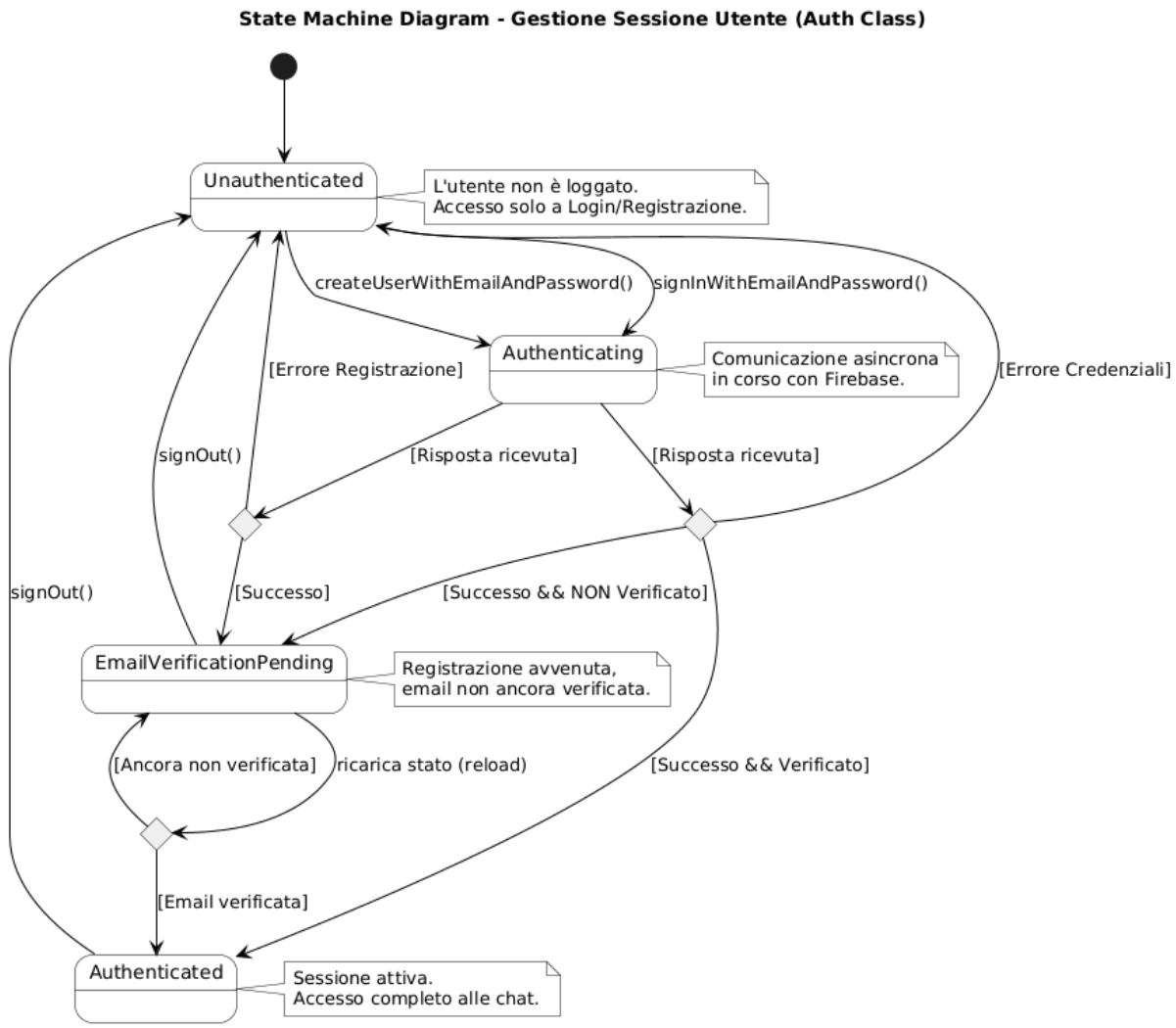
Descrizione dell’interazione tra i vari componenti nel caso in cui l'utente prema un messaggio per leggerlo con la lettura vocale, anche qui è mostrata la divisione architetturale tra Grafica, Logica, Servizio.




#### 4.3.2.2 STATE MACHINE DIAGRAM

**State Machine Diagram: Gestione dello Stato di Autenticazione** Questo diagramma modella il ciclo di vita della sessione utente gestita dalla classe Auth.

1. Stato Iniziale: Unauthenticated.
2. Transizioni:
3. signInWithEmailAndPassword porta allo stato Authenticating.
4. In caso di successo -> Authenticated.
5. In caso di errore -> Ritorno a Unauthenticated con messaggio di errore.
6. Dallo stato Authenticated, l'azione signOut riporta a Unauthenticated.
7. È previsto uno stato intermedio EmailVerificationPending se la mail non è verificata dopo la registrazione.




#### 4.3.2.3 ACTIVITY DIAGRAM

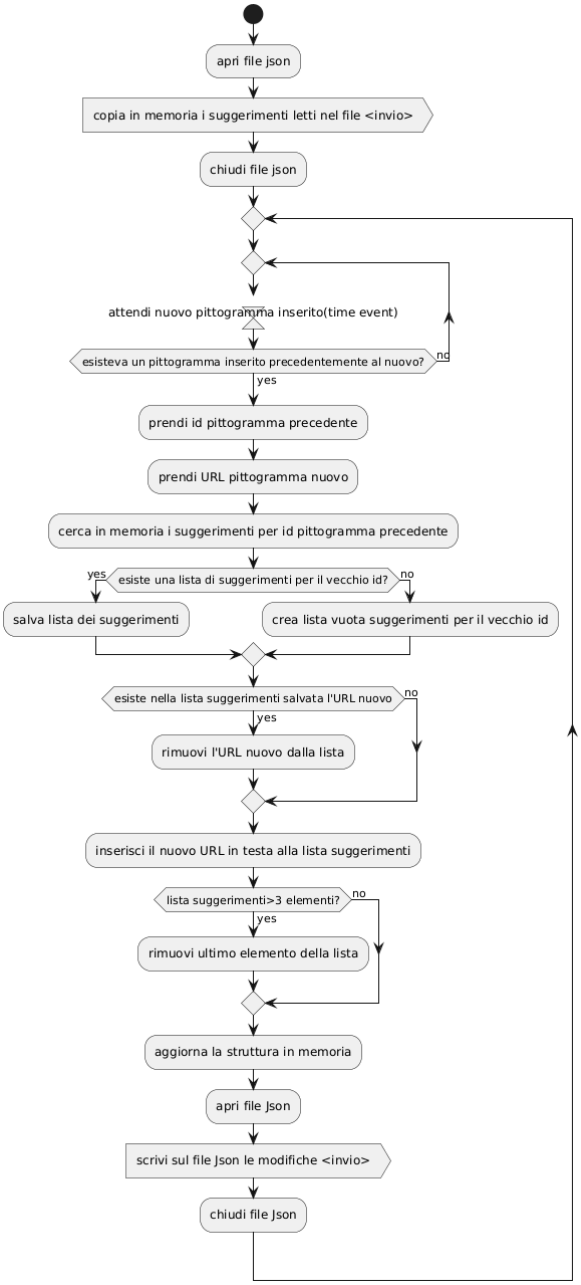
Gestione Mostra Suggerimenti Pittogrammi

Il seguente Activity Diagram descrive l'algoritmo di Gestione dei Suggerimenti dei Pittogrammi, una funzionalità chiave per velocizzare la composizione delle frasi.

Descrizione del Flusso: Inizialmente si leggono al caricamento della pagina i suggerimenti in memoria per avere una maggiore efficienza nella ricerca dei suggerimenti, poi il processo si attiva quando viene inserito un nuovo pittogramma nella frase. Il sistema verifica se esiste un pittogramma precedente per creare un'associazione contestuale.

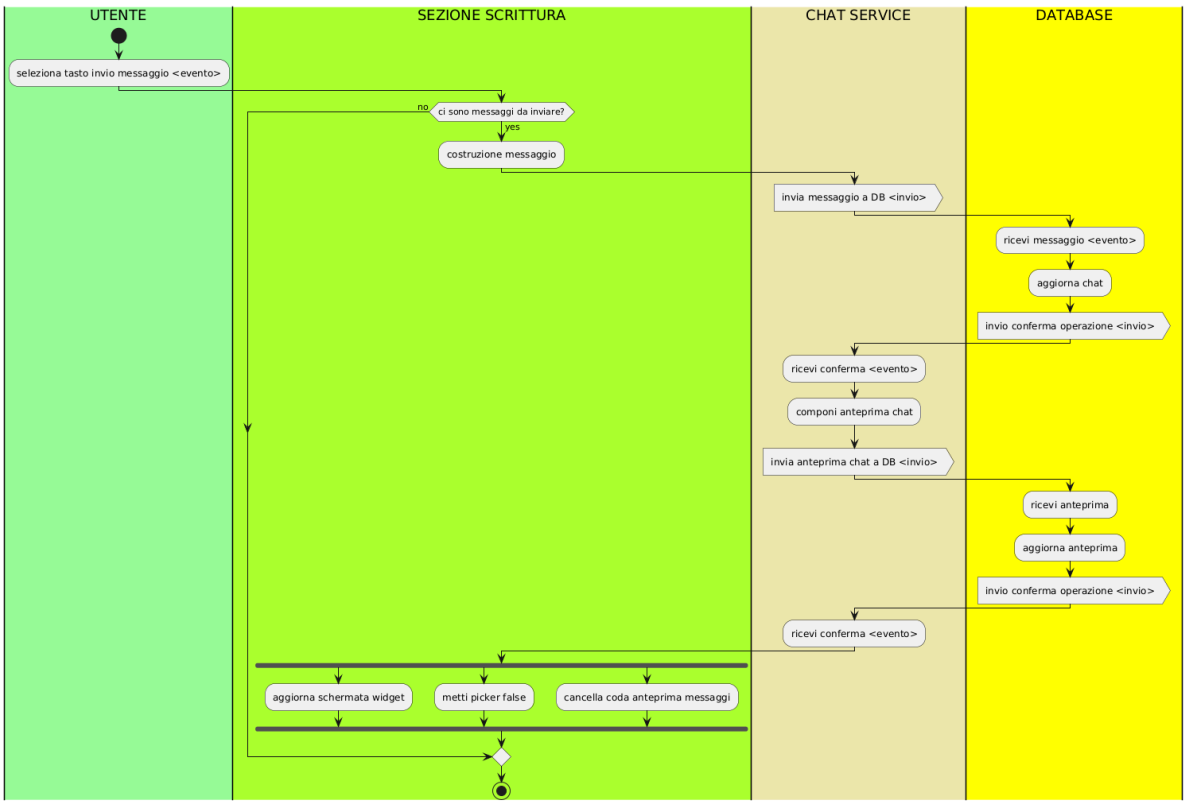
1. Se esiste un pittogramma precedente, il sistema recupera il suo ID e l'URL del nuovo pittogramma.
2. Viene letta la struttura contenente i suggerimenti precedentemente caricati.
3. Se non esiste una lista di suggerimenti per quella coppia, viene creata; altrimenti si aggiorna quella esistente.

4. se la lista supera i 3 elementi, il suggerimento meno recente viene rimosso per mantenere il file leggero.
5. Le modifiche vengono salvate su file JSON locale.



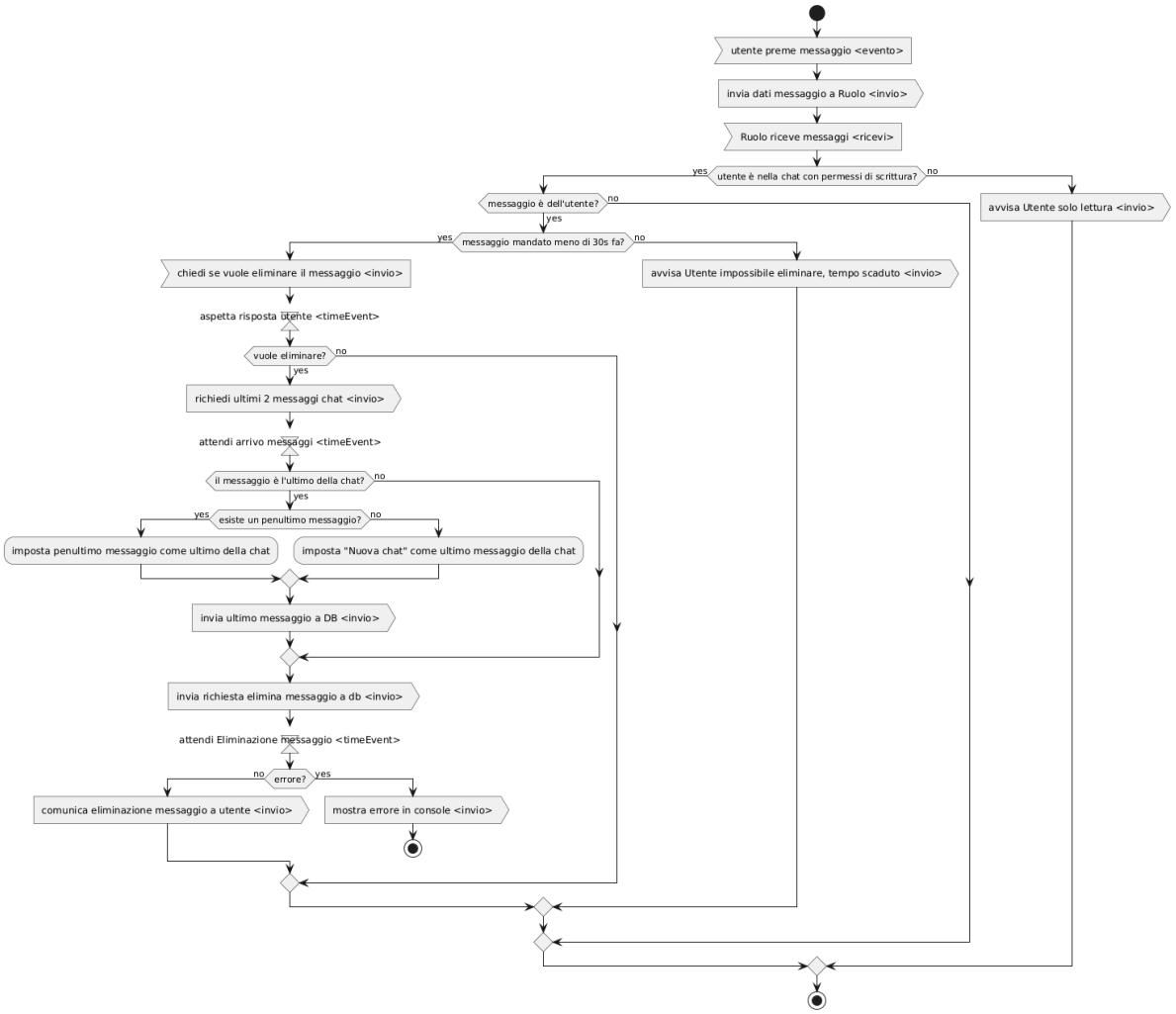

<u>Gestione Invio Messaggio</u>

Descrizione del flusso: il diagramma descrive quale è l'algoritmo da seguire nel caso in cui un utente invia un messaggio descrivendo quali sono le parti interessate nello svolgere la funzionalità di invio messaggio, mostrando così la divisione tra parte front-end(Sezione scrittura) dalla parte back-end(Chat Service) e Database



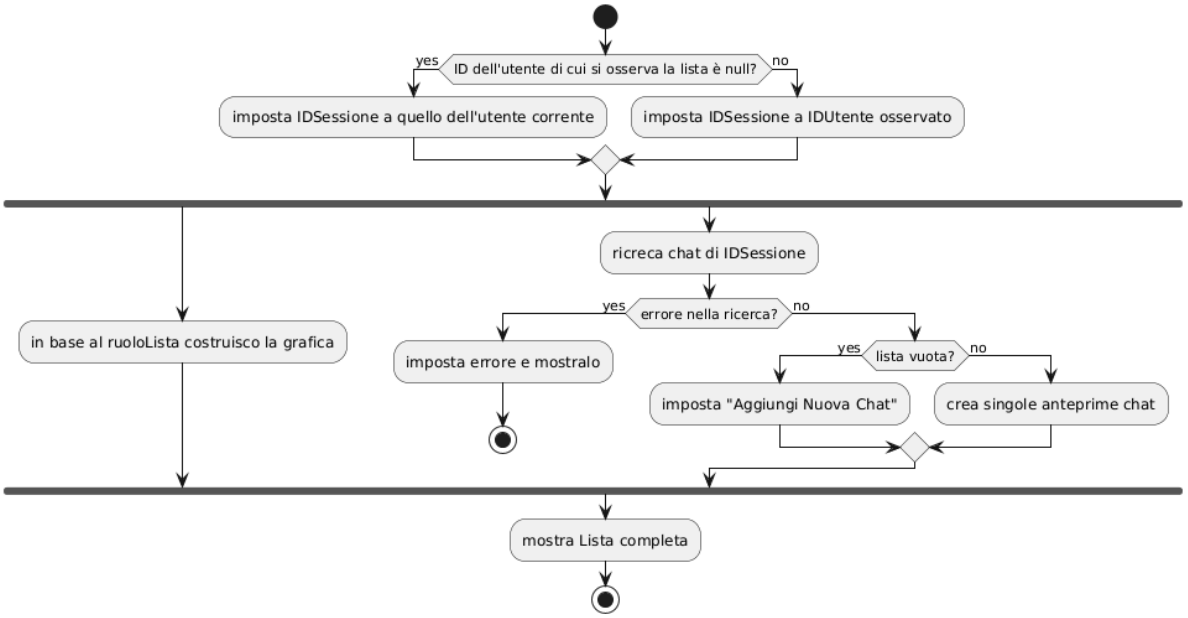

Gestione Eliminazione Messaggio

Descrizione del flusso: il diagramma mostra quale è l'algoritmo da eseguire quando un utente tenta di eliminare un messaggio, mostrando i casi possibili in base a se l'utente si trova nella propria chat oppure nella chat di un utente che sta osservando. Descrive inoltre come gestire l'aggiornamento dell'anteprima messaggio nella chatList.




## Gestione Creazione ChatList

Descrizione del flusso: il diagramma descrive i passaggi necessari per la realizzazione della chatList quando un utente accede alla propria lista oppure alla lista di un utente del quale vuole controllare le chat.

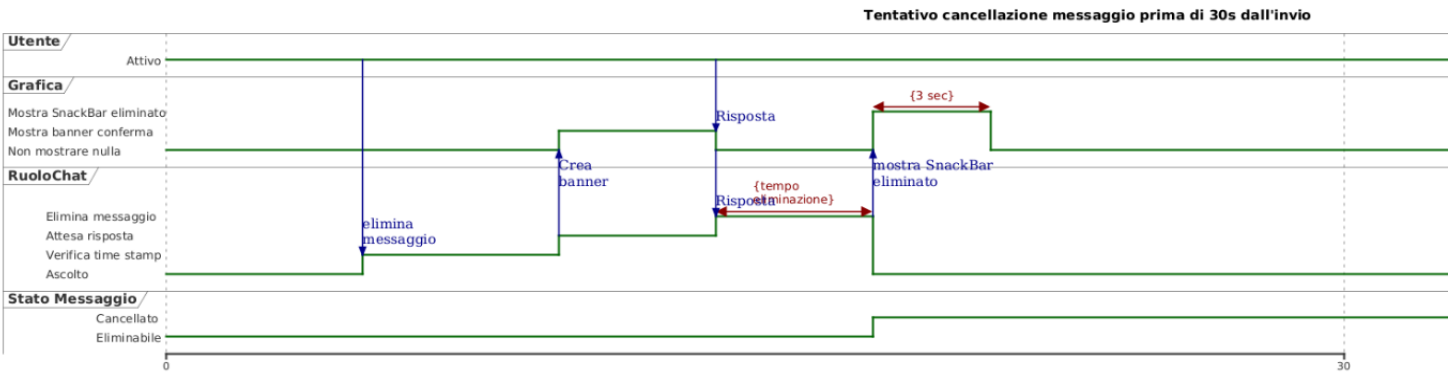


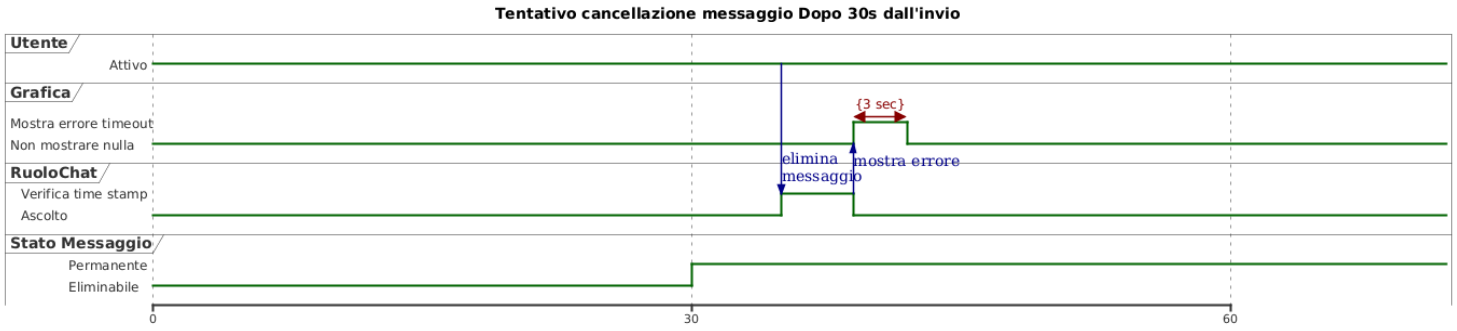

#### 4.3.2.4 TIMING DIAGRAM

<u>Comportamento Eliminazione Messaggio</u>

I seguenti diagrammi temporali mostrano quale deve essere il comportamento dell'applicazione nel caso in cui l'utente tenti di eliminare un messaggio nella chat, se è stato inviato prima di 30 secondi dall'invio allora è possibile eliminarlo (chiedendo la conferma all'utente), altrimenti il sistema avvisa l'impossibilità di eliminare il messaggio.

Il primo diagramma mostra come si deve comportare il sistema quando un utente preme sul messaggio, e se è possibile eliminare o no il messaggio, il secondo mostra un tentativo di eliminazione fallimentare(post 30 secondi) e il terzo un tentativo con successo.




#### 4.3.2.5 USE CASE DIAGRAM

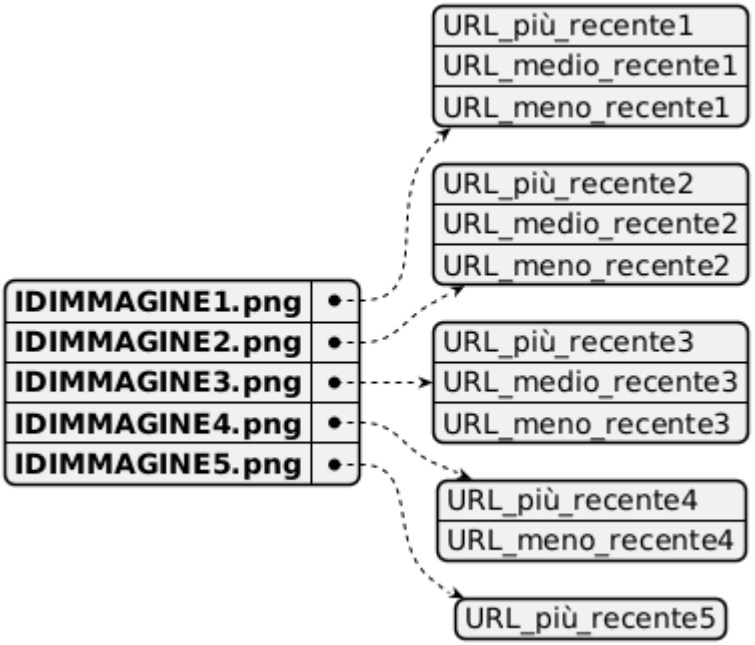
Interazione Utente con ChatPage

Il diagramma mostra quali siano le azioni che un determinato utente può svolgere all'interno della chatPage in base al ruolo assunto, Scrittore (la chat è sua) Osservatore (la chat non è di proprietà dell'utente, l'utente sta leggendo la chat di qualcun altro). Il diagramma mostra anche quali sono gli attori coinvolti per svolgere le azioni.


# 4.3.3 MODELLO STRUTTURE DATI

### 4.3.3.1 STRUTTURA FILE JSON

La struttura del file Json per i singoli pittogrammi è la seguente mostrata da questo diagramma, ogni chiave sarà la parte finale dell'URL del pittogramma contenente l'id dell'immagine e l'estensione del file, ad ogni chiave corrisponderà una lista di elementi da 1 a 3 che corrispondono agli URL dei pittogrammi da suggerire, dal più recente usato al meno




# 4.4 - Design Patterns

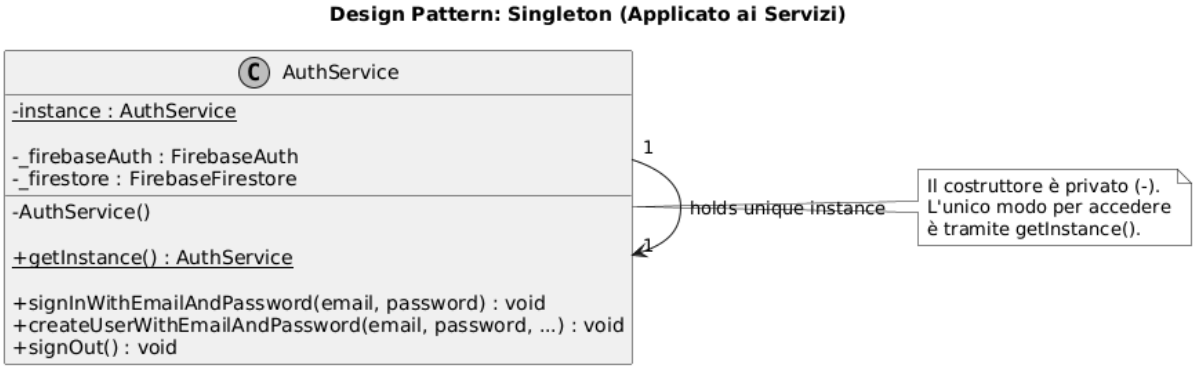
per ingegnerizzare il codice che abbiamo scritto abbiamo deciso di utilizzare i

**1. Singleton Pattern**

1.1 Singleton AuthService

-Contesto**:** È necessario avere un'unica istanza globale per la connessione ai servizi di backend per evitare connessioni multiple e spreco di risorse.

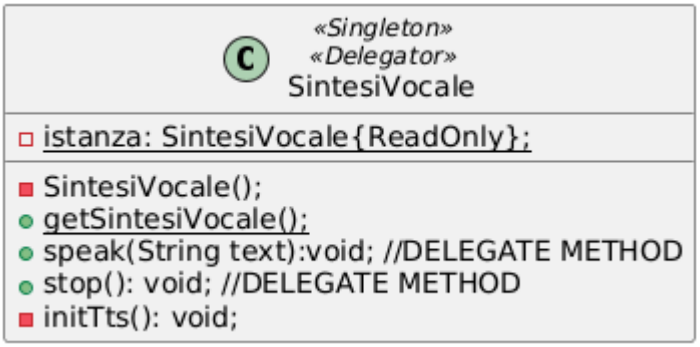
-Applicazione**:** Le classi Auth e ChatService utilizzano le istanze statiche FirebaseAuth.instance e FirebaseFirestore.instance. Anche se incapsulate nei nostri Service, la gestione delle risorse sottostanti segue strettamente il pattern Singleton garantito dall'SDK di Firebase.



1.2 Singleton SintesiVocale

-Contesto: la chat necessita di un servizio di sintesi vocale implementato nella classe SintesiVocale, il servizio inoltre deve essere inizializzato per poter funzionare. La sintesi vocale è un servizio unico per tutte le chat quindi non deve essere possibile la creazione di più istanze(anche per non avere il problema di sovrapposizione di voci) e deve essere inizializzato solo una volta.

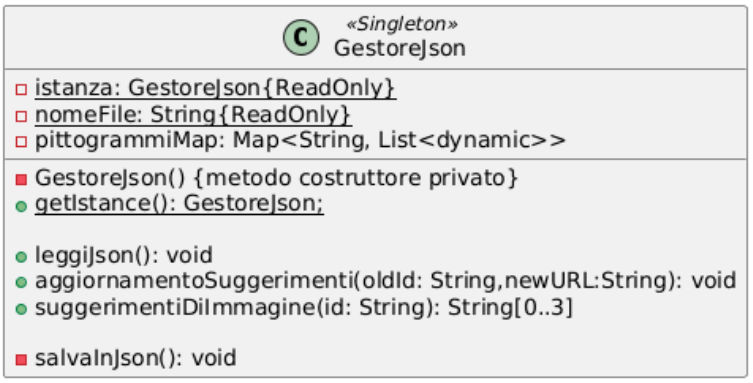
-Applicazione: implementare il pattern Singleton nella classe, questo garantisce che solo in un momento (nel metodo costruttore) venga inizializzato migliorando l'efficienza del codice, ed essendoci sono una istanza della classe la gestione del flusso stop lettura/avvia lettura risulta molto più sicura senza possibilità di sovrapposizione di voci.




1.3 Singleton GestoreJson

-Contesto: è necessaria una classe per la gestione dei suggerimenti per il singolo utente contenuta nel FileSystem dell'utente sotto forma di file Json. La classe deve essere unica poiché esiste solo un gestore del file e non deve esserci la possibilità che più istanze possano modificare il file con il rischio di avere operazioni di scrittura in concomitanza con quelle di lettura(in modo da non accedere a dati incoerenti e non rischiare di corrompere il file).

-Applicazione: usare il pattern Singleton in modo da avere solo una classe autorizzata alla gestione del file e suggerimenti.



1.4 Singleton AvvisatoreSnackBar

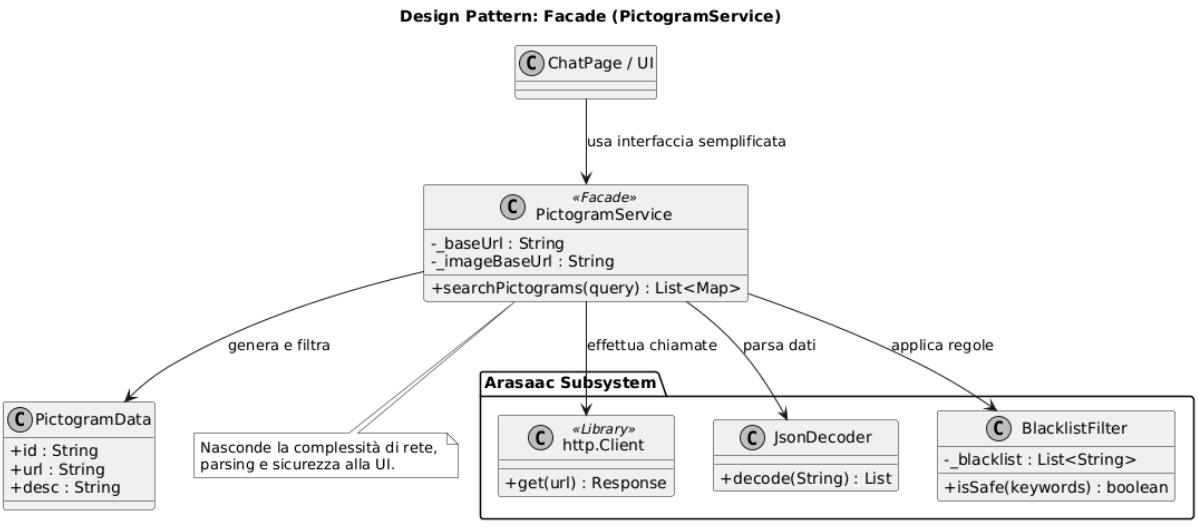
-Contesto: è necessaria una classe che possa permettere la costruzione dei messaggi tramite SnackBar su UI per le comunicazioni con l'utente dell'esito di varie operazioni anche differenti tra loro ma con stile grafico sempre identico senza però avere codice di costruzione dei messaggi duplicato.

-Applicazione: utilizzo del Singleton pattern in modo da avere una classe unica che costruisca l'oggetto SnackBar garantendo una migliore manutenibilità.


## 2. Facade Pattern

-Contesto**:** Il sistema interagisce con sottosistemi complessi (API REST di Arasaac, Database NoSQL Firestore,).

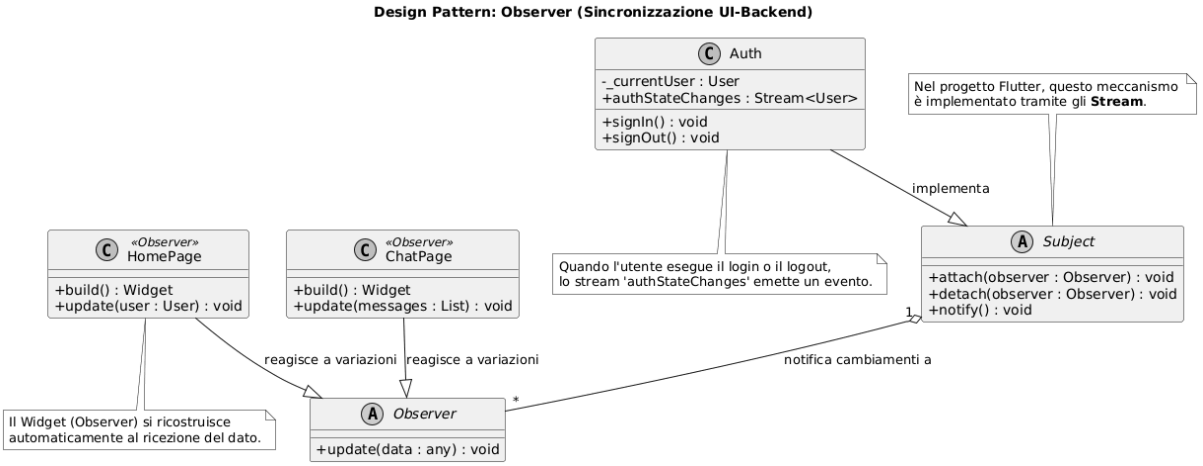
-Applicazione**:** Le classi nel package services (PictogramService, Auth, ChatService) agiscono come **Facade**.



## 3. Observer Pattern

-Contesto: L'applicazione deve essere reattiva: quando arriva un messaggio o cambia lo stato di login, l'interfaccia deve aggiornarsi automaticamente senza polling.

-Applicazione : Utilizziamo gli Stream di Dart. In Auth, il metodo authStateChanges restituisce uno stream che notifica la UI ogni volta che l'utente si logga o slogga. Nel Component Diagram, questo è rappresentato dalla natura "Event-Driven" dell'architettura.

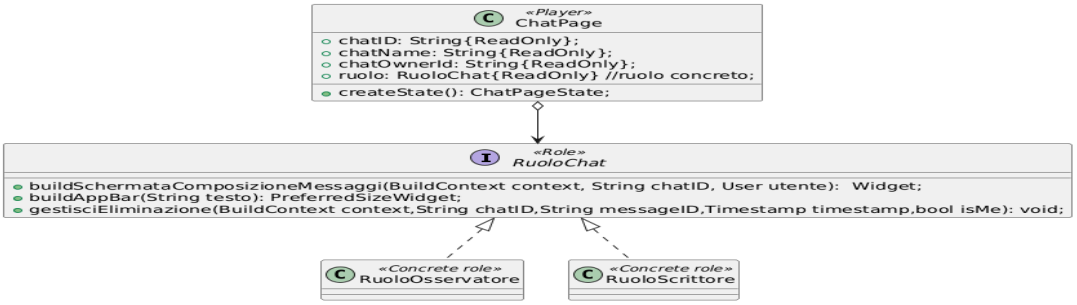



## 4. Player-Role Pattern

4.1 player-role ChatPage:

-Contesto: In base a se il tutor sta osservando una chat del CCN oppure se l'utente sta visualizzando una propria chat, il sistema deve permettere nel primo caso di costruire una chat dove si può solo osservare i messaggi ed essere avvisato che si trova in modalità solo lettura, nel secondo avere la possibilità di eseguire le azioni di invio ed eliminazione dei messaggi.

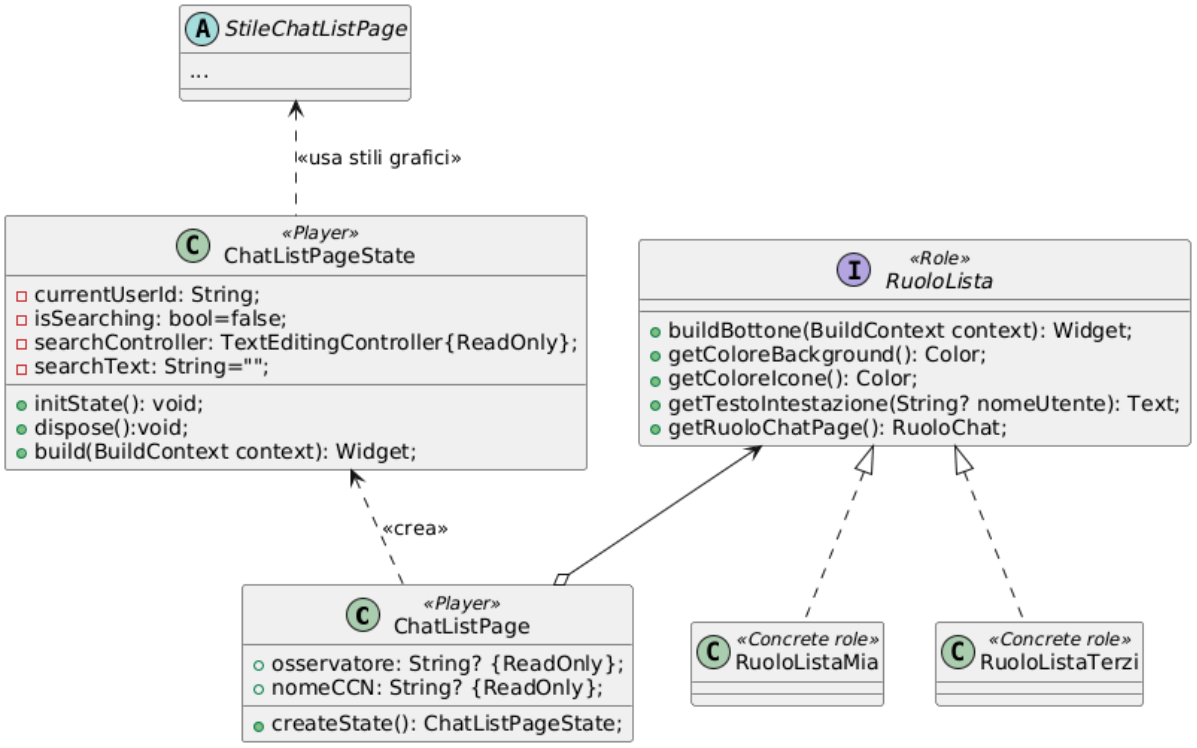
-Applicazione: usiamo due ruoli distinti per la chat, il ruolo Osservatore che permette le azioni del caso 1 e il ruolo Scrittore che permette le azioni del caso 2. in modo da utilizzare lo stesso oggetto chiave (ChatPage) per modellare 2 comportamenti differenti della stessa, aumentando la flessibilità e portabilità del codice(in futuro se si vogliono inserire nuovi ruoli è possibile farlo).




4.2 player-role ChatList:

-Contesto: in base a se il tutor sta accedendo alla lista delle Chat di un CCN oppure se l'utente sta visualizzando la propria lista messaggi: devono essere presenti 2 grafiche differenti per le liste per segnalare la modalità, nel primo caso non deve esserci la possibilità di creare nuove chat a differenza del secondo e in base alla modalità si accede alle singole ChatPage con funzionalità differenti(sopracitate in 4.1)

-Applicazione: usiamo due ruoli distinti per la ChatList, ruolo ListaMia (caso lista propria) e ruolo ListaTerzi (caso lista osservata), così facendo abbiamo la possibilità di utilizzare lo stesso codice in casi differenti ed aumentare la manutenzione futura nel caso in cui si aggiungano ulteriori tipologie di liste.

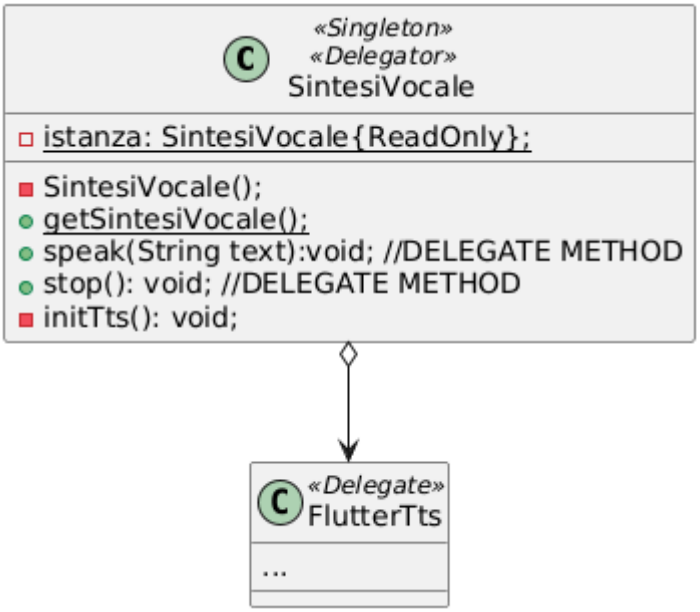


## 5. Delegation Pattern

### 5.1 SintesiVocale

-Contesto: La funzionalità di sintesi vocale utilizza una classe fornita dal una libreria, le operazioni di sintesi però devono svolgere ulteriori operazioni (esempio quando si richiede la lettura di un messaggio prima bisogna fermare una possibile sintesi vocale precedente in corso), inoltre è necessario separare le richieste di lettura dei componenti dall'utilizzo delle chiamate di metodi della classe importata di sintesi per permettere una maggiore manutenibilità del codice nel caso in cui in un futuro si volesse cambiare la classe importata.

-Applicazione: creiamo una classe SintesiVocale che possiede come attributo la classe di sintesi effettiva e nei suoi metodi chiama i vari metodi della classe per performare le operazioni di voce.




### 5.2 GestoreRichiestePreferenze

-Contesto: la gestione di richieste per le preferenze di visualizzazione della chat per gli utenti utilizza il servizio di preferenceService che interagisce con il database per recuperare le preferenze dell'utente restituendo i dati delle preferenze in modo grezzo (mappa), il GestoreRichiestePreferenze deve andare a restituire un oggetto che contiene in modo ordinato le preferenze dell'utente. Inoltre è necessario separare la parte front-end(grafica/UI) dalla parte back-end(Interazione con il server), in altre parole l'interfaccia non deve essere a conoscenza di come vengano recuperate le preferenze, deve solo richiedere i servizi.

-Applicazione: la creazione di una classe GestoreRichiestePreferenze che contiene come attributo la classe preferenceService e nel suo metodo di richiesta di

preferenze va a creare un oggetto apposito contenente le preferenze in modo strutturato(Preferenze chat) che andrà poi ad essere restituito al chiamante del metodo.