



# Progetto SAD

## Web Application Social Notes

- 1 Presentazione del progetto
- 2 Descrizione del progetto
  - 2.1 Processo di sviluppo
  - 2.2 Gestione del team
  - 2.3 Riunione di progetto
  - 2.4 Stima dei costi di sviluppo
  - 2.5 Strumenti utilizzati
  - 2.6 Context Diagram
- 3 Glossario dei termini
- 4 Specifica dei requisiti
  - 4.1 Descrizione testuale dei requisiti
  - 4.2 Descrizione generale
  - 4.3 Descrizioni delle parti di interesse
  - 4.5 Requisiti funzionali
  - 4.6 Requisiti non funzionali
- 5 Analisi dei requisiti
  - 5.1 Identificazione degli attori
  - 5.2 Use Case Diagram
  - 5.3 Descrizione breve dei casi d'uso
  - 5.4 Descrizione in dettaglio dei casi d'uso
  - 5.5 Modello di Dominio
  - 5.6 Diagramma di Sequenza di analisi
  - 5.6 Activity Diagram
- 6 Architettura software
  - 6.1 Viste architettoniche di alto livello
  - 6.2 Component diagram
    - 6.2.1 Diagramma dei componenti dettagliato
    - 6.2.2 Diagrammi delle interfacce Server-Database
  - 6.3 Descrizione dell'architettura di dettaglio
- 7 Persistenza dei dati
  - 7.1 Scelta del DBMS
  - 7.2 Individuazione delle Collection
  - 7.3 Costruzione del database
- 8 Implementazione
  - 8.1 Deployment Diagram
- 9 Testing
  - 9.1 **Testing di Unità**
  - 9.2 **Testing di Integrazione**
  - 9.3 **Testing di interfaccia**
- 10 Sviluppi futuri

# Web Application Social Notes

## 1 Presentazione del progetto

Si intende sviluppare una **Web Application** per la **condivisione di materiale accademico** per una *cerchia di studenti* sotto forma di post. Gli utenti consultano e commentano degli appunti riguardanti un oggetto di studio.

Il sistema che verrà realizzato prevede la possibilità di registrarsi sulla piattaforma, eseguire l'accesso, in modo tale tale da: pubblicare, valutare e commentare materiale nuovo o già esistente sulla piattaforma.

Ogni utente può commentare il materiale esistente ed eliminare eventualmente un commento creato da egli stesso, come anche eliminare un post di sua "proprietà".

Il nostro obiettivo è quello di proporre un'applicazione utile e funzionale ma allo stesso tempo accattivante ponendola nelle vesti di un "social", tecnologia di tendenza nell'ultimo decennio(da questo il nome "social" "Notes").

## 2 Descrizione del progetto

Per lo sviluppo sono state adottate metodologie di sviluppo agile in quanto si è scelto un approccio di tipo **iterativo** ed **evolutivo**, con iterazioni brevi e **timeboxed** e una pianificazione adattiva.

### 2.1 Processo di sviluppo

Si è utilizzato più nello specifico un processo di sviluppo software **UP** (Unified Process) in cui c'è stata una prima fase di **Ideazione** in cui si è effettuato uno studio di fattibilità con le stime dei costi e dei tempi di sviluppo.(riportati nel paragrafo 2.3 e 2.4)

Successivamente si è passati alla fase di **Elaborazione**, in cui attraverso un'implementazione iterativa che consta di due iterazioni si è implementato il nucleo dell'architettura, andando ad implementare i casi d'uso di priorità maggiore.

Dopo di che si è passati alla fase di **Costruzione** che consta di una iterazione in cui si sono implementati i casi d'uso rimanenti, a minor rischio.

Di seguito sono riportati i documenti di Implementazione:

Implementazione	Inizio	Fine	Durata
<b>Iterazione 1</b>	5 dicembre	25 dicembre	20 giorni
Esplorazione dei requisiti	5 dicembre	9 dicembre	
Implementazione Server Spring Boot	10 dicembre	25 dicembre	
<b>Iterazione 2</b>	26 dicembre	8 gennaio	14 giorni
Implementazione base del Client in Angular	26 dicembre	30 dicembre	
Implementazione caso d'uso LOGIN e REGISTRA, Pubblica Post	31 dicembre	8 gennaio	
<b>Iterazione 3</b>	11 gennaio	24 gennaio	13 giorni
Implementazione caso d'uso Visualizza Post, Cerca Post, Elimina Post.	11 gennaio	20 gennaio	
Implementazione caso d'uso Commenta Post, Valuta Post, Elimina Commento	16 gennaio	21 gennaio	

Ed infine si è passati alla fase di **Transizione** in cui è presente il Beta Test e il Rilascio dell'applicazione.(rispettivamente i paragrafi 8 e 9 )

Sono state adottate le pratiche più comuni del processo di sviluppo UP come :

- Affrontare i problemi da rischio maggiore nelle iterazioni iniziali.
- Testing frequente, presto e realistico degli output di ogni iterazione.

- Modellazione visuale con UML con le opportune discipline.
- Applicazione dei casi d'uso.
- Coinvolgimento del cliente (la professoressa Anna Rita Fasolino) sulla valutazione e feedback, ed inoltre anche per le richieste di cambiamento del software o l'aggiunta di funzionalità.

## 2.2 Gestione del team

Il team è composto da tre membri, ogni fase di sviluppo è stata suddivisa in task e ad ogni membro del gruppo è stato assegnato uno o più di questi task. Al termine di ogni ciclo, attraverso una riunione di tutto il team, si è discusso dei risultati ottenuti e si sono individuati i prossimi task da completare.

Tutto questo è stato favorito dall'utilizzo della piattaforma **Notion** per la coordinazione di scadenze, incarichi e riepilogo obiettivi. È stata utilizzata anche la piattaforma **Microsoft Teams**, necessaria per permettere al team di lavorare anche in remoto.

## 2.3 Riunione di progetto

Lo sviluppo ha richiesto circa 45 giorni, con un impegno individuale per membro di 6/7 giorni a settimana.

Attività svolta	Sett. 1	Sett. 2	Sett. 3	Sett. 4	Sett. 5	Sett. 6	Totale ore
Studio di fattibilità	20						20
Analisi Requisiti	10	10		5			25
Progettazione	20	20	30	30			100
Implementazione		10	20	20	30	10	90
Test			10		10	20	40
Gestione Documentazione	5	10	5		10	20	50
Total ORE	55	50	65	55	50	50	325

## 2.4 Stima dei costi di sviluppo

Il calcolo dei costi di sviluppo del progetto è stato basato sulla tecnica degli **Use Case Point**. Dopo un'attenta analisi sui possibili use-case del progetto si è fatta una stima dei differenti indici di stima: come prima approssimazione si è proceduto al calcolo dell'indicatore **UUCP(Unadjusted Use Case Point)** :

$$\text{UUCP} = \text{UUCW} + \text{UAW} = 120 + 7 = 127 \quad [\text{Tabella 2.6A e Tabella 2.6B}]$$

Successivamente sono stati calcolati il fatto di complessità tecnica TFC e il fattore ambientale EF:

$$\text{TFC} = 0.6 + (0.01 * \text{TFactor}) = 0.6 * (0.01 * 42) = 0.25 \quad [\text{Tabella 2.6C}]$$

$$\text{EF} = 1.4 + (-0.03 * \text{EFactor}) = 1.4 + (-0.03 * 30.5) = 0.49 \quad [\text{Tabella 2.6D}]$$

Moltiplicando gli indicatori calcolati si ottiene l'indicatore UCP(Use Case Point):

$$\text{UCP} = \text{UUCP} * \text{TFC} * \text{EF} = 15,55$$

Supponendo un lavoro di 25 ore per ogni Use Case Point si ottiene un quantitativo di ore pari a:

$$\text{Ore} = 25 * 15,55 = 388,75\text{H}$$

Complessità	Peso	Numeri di casi d'uso	Prodotto
Semplice	5	0	Text
Medio	10	3	30
Complesso	15	6	90
Totale		9	<b>120</b>

**Tabella 2.6A**

Tipoogia Attore	3	Numeri Attori	Prodotto
Semplice	1	1	1
Medio	2	0	0
Complesso	3	2	6
Totale		3	<b>7</b>

**Tabella 2.6B**

Fattore	Descrizione	Peso	Valutazione	Prodotto
T1	Sistema distribuito	2.	3	6
T2	Performance	2	3	6
T3	Efficienza per l'utente	1	3	3
T4	Complessità di elaborazione interna	1	2	2
T5	Riusabilità del codice	1	0	0
T6	Facile da installare	0.5	0	0
T7	Facile da usare	0.5	4	2
T8	Portable	2	2	4
T9	Facile da cambiare	1	5	5
T10	Elaborazione parallela	1	5	5
T11	Caratteristiche di sicurezza	1	5	5
T12	Accesso diretto da terzi	1	3	3
T13	Formazione per utenti finali	1	0	1
<b>Totale</b>				<b>42</b>

**Tabella 2.6C**

Fattore	Descrizione	Peso	Valutazione	Prodotto
E1	Familiarità con il processo di sviluppo	1.5	4	6
E2	Esperienza di applicazione	0.5	3	1.5
E3	Esperienza Object-oriented del team	1	5	5
E4	Capacità di analisi	0.5	4	2
E5	Motivazione del team	1	5	5
E6	Stabilità dei requisiti	2	3	6
E7	Personale part-time	-1	1	-1
E8	Complessità del linguaggio utilizzato	2	3	6
<b>Totale</b>				<b>30.5</b>

**Tabella 2.6D**

## 2.5 Strumenti utilizzati



È servizio di database multi-cloud creato per la resilienza, la scalabilità e i massimi livelli di privacy e sicurezza dei dati. In particolare è un **DBMS non relazionale**, orientato ai documenti.



È un **framework** open source per lo sviluppo di **applicazioni web** con Licenza MIT.



IntelliJ IDEA è un **ambiente di sviluppo integrato** per il linguaggio di programmazione **Java**. Sviluppato da JetBrains, è disponibile sia in licenza Apache che in edizione proprietaria commerciale.



Microsoft Teams è una **piattaforma di comunicazione** aziendale proprietaria sviluppata da Microsoft, come parte della famiglia di prodotti Microsoft 365.



In informatica la piattaforma Java è una **piattaforma software**, sviluppata su specifiche e implementazioni di Sun Microsystems ovvero l'ambiente di esecuzione necessario per l'esecuzione di programmi scritti in linguaggio java.



Spring è un **framework** open source per lo sviluppo di **applicazioni su piattaforma Java**.

Il progetto Spring Data MongoDB fornisce l'**integrazione** con il database dei documenti **MongoDB**. Le aree funzionali chiave di Spring Data MongoDB sono un modello incentrato su POJO per interagire con una DBCollection MongoDB e scrivere facilmente un livello di accesso ai dati in stile repository.



GitHub, Inc. è un **servizio di hosting** Internet per lo **sviluppo di software** e il controllo della versione tramite Git.



Draw.io è un **software di disegno grafico** multipiattaforma gratuito e open source sviluppato in HTML5 e JavaScript. La sua interfaccia può essere utilizzata per creare diagrammi come diagrammi di flusso, wireframe, diagrammi UML, organigrammi e

diagrammi di rete.



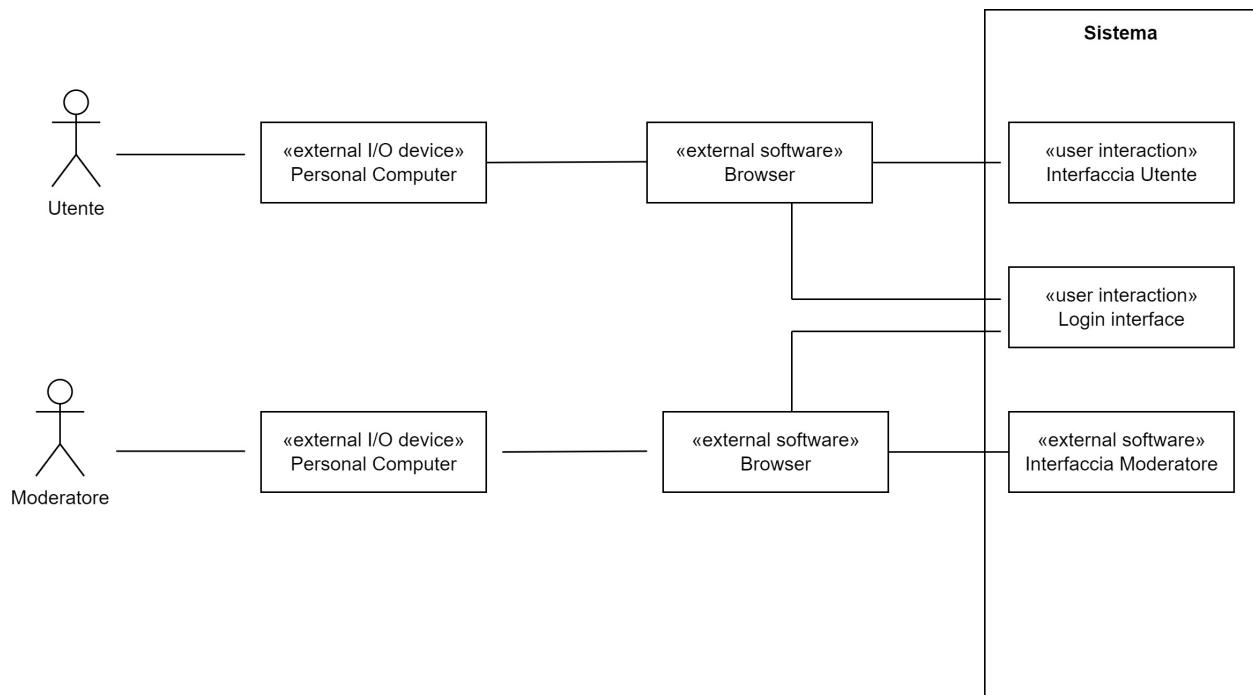
Notion è un programma per **prendere appunti** e gestire i **progetti**, che aiuta gli utenti a coordinare **scadenze, obiettivi e incarichi** per una maggiore efficienza e produttività.

## 2.6 Context Diagram

Identificazione degli attori:

- Utente
- Moderatore

Gli attori attraverso un personal computer sul quale `e installato un browser possono accedere al sistema. Il browser consente di effettuare richieste HTTP al web server per richiedere le interfacce con le quali gli attori possono accedere ai servizi offerti dal sistema.



## 3 Glossario dei termini

Termine	Definizione	Regole di validazione
Utente	Utilizzatore dell'applicativo, il quale consulta e pubblica i post.	Deve essere loggato alla piattaforma

Utente Non Registrato	Visitatore che non possiede ancora una registrazione all'interno del sistema.	
Moderatore	Utente speciale che ha la responsabilità di supervisionare le politiche di comportamentali sulla piattaforma	Deve essere loggato alla piattaforma
Post	Unità atomica di condivisione del materiale per l'applicativo. Può essere commentato, valutato, eliminato e segnalato.	E' identificato da un codice univoco
Valutazione di un Post	Voto dato da un utente, compreso in un range da 1 a 30.	
Commento	Frase testuale inherente ad un post	E' identificato da un codice univoco

## 4 Specifica dei requisiti

### 4.1 Descrizione testuale dei requisiti

Si vuole realizzare un sistema software per la gestione della condivisione di materiale accademico tra utenti iscritti sulla piattaforma. Un utente , che si identifica mediante Username e Password può inserire sulla piattaforma uno o più post che trattano di appunti che possono essere visualizzati e commentati dagli altri utenti della piattaforma. Per favorire una ricerca più efficace del materiale di interesse sono state aggiunte dei filtri per categoria e una barra di ricerca per parole/ frasi chiave.

Infine è possibile eliminare i propri commenti che sono stati scritti sotto ad un post ed eliminare il post stesso pubblicato laddove vi sia questa volontà o necessità.

### 4.2 Descrizione generale

#### 4.2.1 Prospettive del prodotto

Lo scopo di questa piattaforma è quello di gestire tutto ciò che concerne la condivisione di materiale accademico avvenuta tramite la pubblicazione di post. Ciò aiuterebbe gli utenti ad ampliare la loro preparazione per un esami, semplificando il reperimento di materiale e può essere da stimolo per la nascita di nuove amicizie e legami.

#### 4.2.2 Opportunità di business

La web application si pone come elemento di connessione tra gli studenti e non che decidono di iscriversi alla piattaforma, favorendo lo scambio di informazioni tra quest'ultimi in modo facile ed intuitivo. In futuro questo prodotto potrebbe essere ampliato, introducendo dei corsi, delle challenge e dei premi per aumentare il grado di attività sulla piattaforma portandola ad essere a tutti gli effetti uno strumento educativo.

#### 4.2.3 Demografia di mercato

Il prodotto è destinato a tutte le persone che sono alla ricerca o che posseggono del materiale accademico da condividere principalmente con gli altri colleghi. Con un solo strumento quest'ultimi saranno interconnessi e potranno scambiarsi delle informazioni come utenti della piattaforma.

### 4.3 Descrizioni delle parti di interesse

- **Utente:** entità registrata sulla piattaforma che in cerca di materiale accademico di interesse e che condivide quello in suo possesso.

- **Utente non registrato:** tale entità dovrebbe avere la possibilità di registrarsi sulla piattaforma.
- **Moderatore :** entità registrata sulla piattaforma che ha il compito di supervisionare i post e i commenti segnalati per procedere eventualmente alla loro eliminazione definitiva.

## 4.5 Requisiti funzionali

### UTENTE REGISTRATO

Un utente dopo essersi collegato alla piattaforma può usufruire di una serie di funzionalità che gli permettono di:

- Effettuare il login
- Ricercare un post
- Pubblicare un post
- Consultare un post
- Commentare un post
- Eliminare un post
- Valutare un post
- Segnalare un post

### UTENTE NON REGISTRATO

L'utente non registrato può:

- Registrarsi sulla piattaforma

## 4.6 Requisiti non funzionali

I requisiti non funzionali del nostro progetto sono:

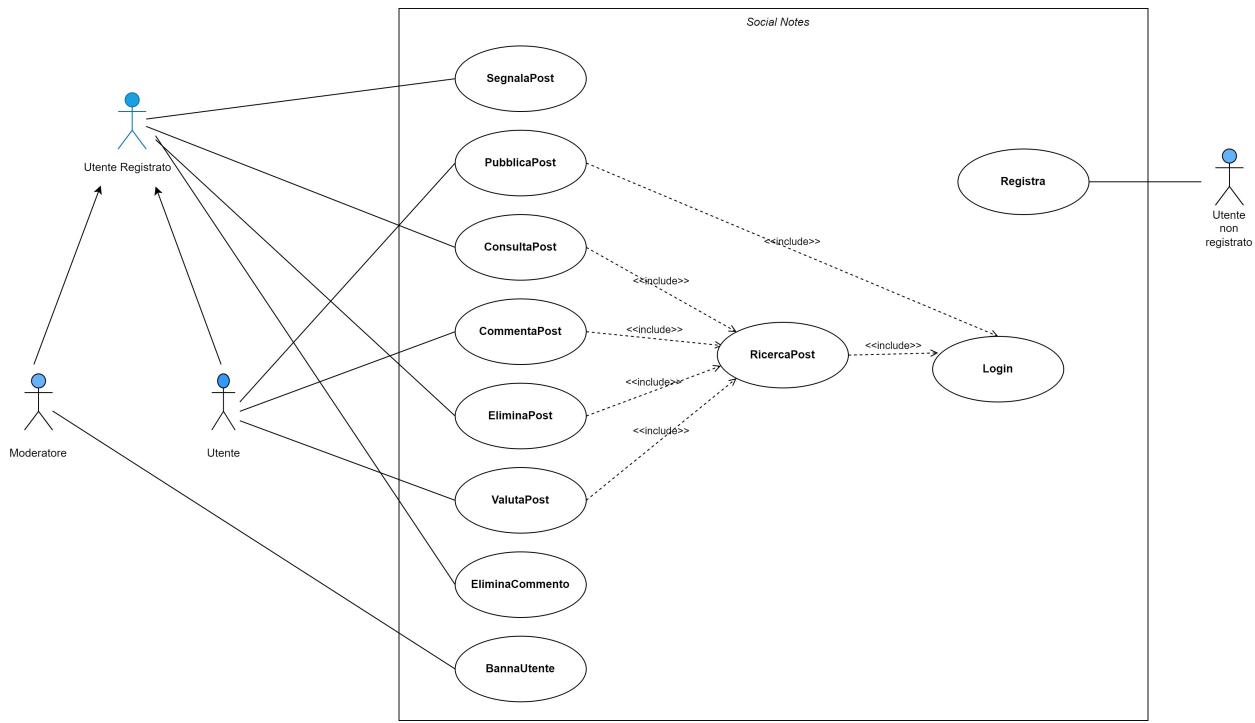
- **Sicurezza:** È necessario che il sistema consenta all'Utente di effettuare le sole operazioni per le quali è autorizzato.
- **Usabilità:** È necessario che Studenti e Moderatori siano in grado di visualizzare in modo chiaro le schermate dedicate, e che possano compiere sul sistema le azioni di competenza con velocità e facilità.
- **Affidabilità:** È necessario garantire la persistenza dei dati su Sistemi Esteri, cosicché sia possibile eseguirne recupero e ripristino anche in caso di fallimento dell'applicativo.
- **Performance:** È necessario che l'esperienza di utilizzo dell'applicativo sia fluida per tutte le tipologie di Utente.
- **Manutenibilità:** È necessario che l'applicativo sia facilmente modificabile, al fine di apportare correzioni, migliorie, adattamenti e aggiunte di funzionalità. A tal fine si prevede che il codice sia quanto più leggibile, così da agevolare le attività di localizzazione delle modifiche e degli errori.

## 5 Analisi dei requisiti

### 5.1 Identificazione degli attori

- **Utente Registrato**
- **Utente Non Registrato**
- **Moderatore**

### 5.2 Use Case Diagram



La frecce tra gli Attori : Utente Registrato , Utente e Moderatore stanno ad indicare una relazione di **generalizzazione**.

Gli attori specializzati, Utente e Moderatore:

- 1) "ereditano" la partecipazione a tutti i casi d'uso con i quali comunica l'attore Utente Registrato.
- 2) possono partecipare ad ulteriori casi d'uso ai quali l'attore Utente Registrato non è collegato.

### 5.3 Descrizione breve dei casi d'uso

- **PubblicaPost:** L'utente pubblica il proprio materiale sulla piattaforma sotto forma di post.
- **ConsultaPost:** L'utente dopo aver ricercato il post lo consulta.
- **CommentaPost:** L'utente dopo aver ricercato il post inserisce un commento testuale sotto quest'ultimo.
- **EliminaPost:** L'utente dopo aver ricercato il post lo elimina.
- **ValutaPost:** L'utente dopo aver ricercato il post effettua una valutazione di quest'ultimo.
- **Registra:** L'utente effettua la registrazione sulla piattaforma.
- **Login:** L'utente si autentifica sulla piattaforma attraverso nome utente e password.
- **RicercaPost:** L'utente ricerca attraverso delle parole chiave uno specifico post (es. materia).
- **EliminaCommento:** L'utente elimina un commento che ha scritto.

### 5.4 Descrizione in dettaglio dei casi d'uso

#### PUBBLICA POST

NOME	PubblicaPost
PORTATA	Notes
LIVELLO	Obiettivo Utente

NOME	PubblicaPost
ATTORE PRIMARIO	Utente registrato
PARTI INTERESSATE e INTERESSEI	Utente che vuole aggiungere nuovo materiale alla piattaforma
PRE-CONDIZIONI	L'utente deve essere loggato
GARANZIA DI SUCCESSO	L'utente può visualizzare il suo post appena pubblicato
SCENARIO PRINCIPALE	<b>1.</b> L'Utente carica tutti i file che vuole pubblicare <b>2.</b> L'utente scrive una descrizione del materiale caricato <b>3.</b> Il sistema controlla che i dati inseriti siano corretti <b>4.</b> Il sistema aggiunge il post al database
ESTENSIONI	Se i dati inseriti non sono validi <b>1a.</b> Il sistema risponde con un messaggio di errore

#### LOGIN

NOME	Login
PORTATA	Notes
LIVELLO	Obiettivo Utente
ATTORE PRIMARIO	Utente registrato
PARTI INTERESSATE e INTERESSEI	Utente che si vuole loggare sul sistema
PRE-CONDIZIONI	L'utente deve essere registrato
GARANZIA DI SUCCESSO	L'utente può accedere alla sua area personale
SCENARIO PRINCIPALE	<b>1.</b> L'Utente inserisce il suo nome utente e la password <b>2.</b> Il sistema controlla che i dati inseriti siano corretti <b>3.</b> L'utente accede alla sua area personale
ESTENSIONI	Se i dati inseriti non sono validi <b>1a.</b> Il sistema risponde con un messaggio di errore

#### REGISTRA

NOME	Registra
PORTATA	Notes
LIVELLO	Obiettivo Utente
ATTORE PRIMARIO	Utente non registrato
PARTI INTERESSATE e INTERESSEI	Utente che vuole registrarsi sulla piattaforma
PRE-CONDIZIONI	L'utente non deve già essere registrato
GARANZIA DI SUCCESSO	L'utente può loggarsi sulla piattaforma con i dati inseriti
SCENARIO PRINCIPALE	<b>1.</b> L'Utente inserisce i suoi dati personali <b>2.</b> Il sistema controlla che i dati inseriti siano corretti <b>3.</b> Il sistema registra l'utente
ESTENSIONI	Se i dati inseriti non sono validi <b>1a.</b> Il sistema risponde con un messaggio di errore

#### COMMENTA POST

NOME	Commenta Post
PORTATA	Notes
LIVELLO	Obiettivo Utente
ATTORE PRIMARIO	Utente registrato
PARTI INTERESSATE e INTERESSEI	Utente che vuole commentare un post
PRE-CONDIZIONI	L'utente deve essere loggato
GARANZIA DI SUCCESSO	L'utente può rileggere il suo commento sotto il post
SCENARIO PRINCIPALE	<b>1.</b> L'utente scrive il testo del commento <b>2.</b> L'utente preme il tasto commento per salvare sulla piattaforma il proprio commento

NOME	Commenta Post
ESTENSIONI	

#### VALUTA POST

NOME	Valuta Post
PORTATA	Notes
LIVELLO	Obiettivo Utente
ATTORE PRIMARIO	Utente registrato
PARTI INTERESSATE e INTERESSEI	Utente che vuole valutare un post
PRE-CONDIZIONI	L'utente deve essere loggato L'utente non deve aver già valutato il post
GARANZIA DI SUCCESSO	L'utente può vedere la valutazione del post aggiornarsi
SCENARIO PRINCIPALE	<b>1.</b> L'utente scrive la valutazione da voler dare al post <b>2.</b> Il sistema salva la valutazione e aggiorna quella del post
ESTENSIONI	Se l'utente ha già valutato il post <b>1a.</b> L'utente non può valutare di nuovo

#### ELIMINA COMMENTO

NOME	Elimina Commento
PORTATA	Notes
LIVELLO	Obiettivo Utente
ATTORE PRIMARIO	Utente registrato
PARTI INTERESSATE e INTERESSEI	Utente che vuole eliminare un suo commento sotto a un post
PRE-CONDIZIONI	L'utente deve essere loggato L'utente deve aver commentato in precedenza il post
GARANZIA DI SUCCESSO	L'utente può vedere il suo commento eliminato
SCENARIO PRINCIPALE	<b>1.</b> L'utente preme il bottone "elimina" vicino al suo commento <b>2.</b> Il sistema provvede ad eliminare il commento
ESTENSIONI	Se l'utente non ha commentato il post <b>1.</b> L'utente non può eliminare commenti scritti da altri utenti

#### RICERCA POST

NOME	Ricerca Post
PORTATA	Notes
LIVELLO	Obiettivo Utente
ATTORE PRIMARIO	Utente registrato
PARTI INTERESSATE e INTERESSEI	Utente che vuole cercare un post specifico
PRE-CONDIZIONI	L'utente deve essere loggato
GARANZIA DI SUCCESSO	L'utente può vedere la pagina home aggioranarsi con i risultati di ricerca
SCENARIO PRINCIPALE	<b>1.</b> L'utente scrive una stringa di caratteri per la ricerca. <b>2.</b> L'utente può selezionare una categoria per mostrare un elenco di post filtrati <b>3.</b> Il sistema ricerca i post in base al testo o alla categoria
ESTENSIONI	Se nel sistema non ci sono post che soddisfano i requisiti di ricerca: <b>1.</b> Il sistema mostrerà a schermo una pagina vuota

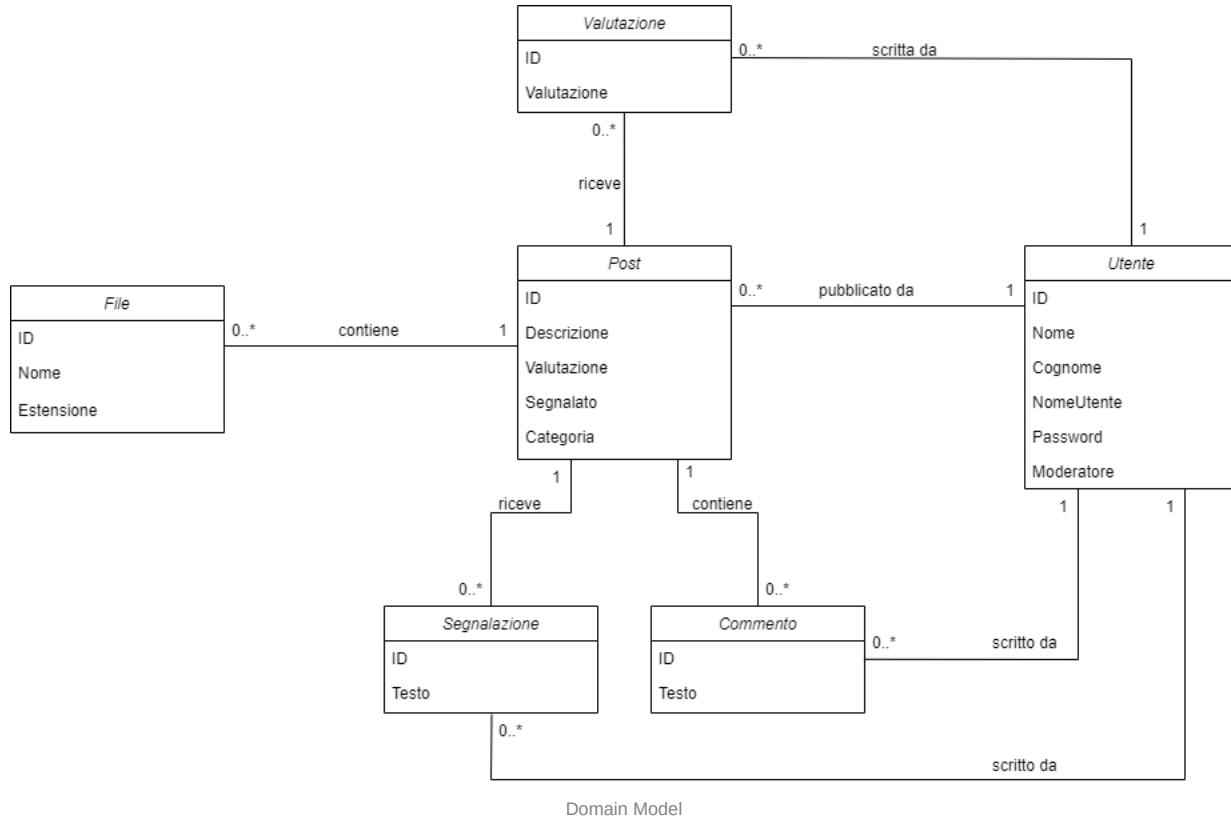
#### ELIMINA POST

NOME	Elimina Post
PORTATA	Notes

NOME	Elimina Post
LIVELLO	Obiettivo Utente
ATTORE PRIMARIO	Utente registrato
PARTI INTERESSATE e INTERESSI	Utente che vuole eliminare un proprio post
PRE-CONDIZIONI	L'utente deve essere loggato L'utente deve essere il creatore del post
GARANZIA DI SUCCESSO	L'utente può vedere il suo post eliminato
SCENARIO PRINCIPALE	<b>1.</b> L'utente preme il bottone "elimina" vicino al suo post <b>2.</b> Il sistema elimina il post
ESTENSIONI	Se l'utente non è il creatore del post che vuole eliminare: 1a. Il sistema non permette di visualizzare il bottone "elimina" su post non propri.

## 5.5 Modello di Dominio

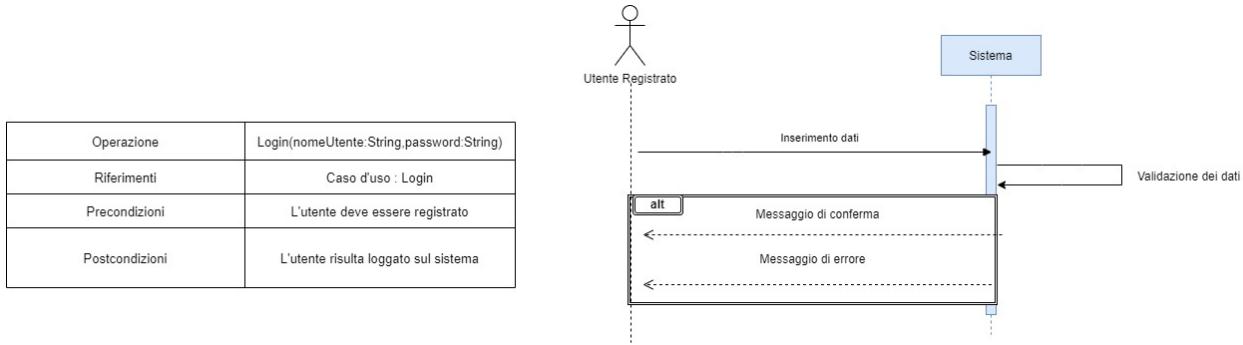
Di seguito è riportato il diagramma di dominio, che descrive le varie entità che fanno parte o hanno rilevanza nel sistema e le loro relazioni.



## 5.6 Diagramma di Sequenza di analisi

Di seguito sono riportati i diagrammi di sequenza, utili per visualizzare le interazioni tra un attore e il sistema. Questo diagramma mette in luce una determinata sequenza di azioni in cui tutte le scelte sono state effettuate.

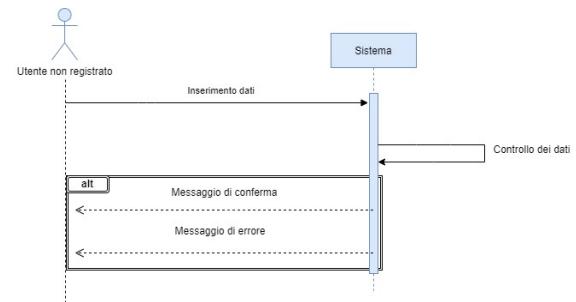
### LOGIN



## REGISTRAZIONE

**Utente non registrato**

Operazione	Registrazione(nome:String,cognome:String,nomeUtente:String,email:String,password String)
Riferimenti	Caso d'uso : Registrazione
Precondizioni	L'utente non deve essere registrato
Postcondizioni	<ul style="list-style-type: none"> <li>Viene creata un'istanza di utente U</li> <li>U.nome viene sostituito da nome</li> <li>U.cognome viene sostituito da cognome</li> <li>U.email viene sostituito da email</li> <li>U.password viene sostituito da password</li> </ul>

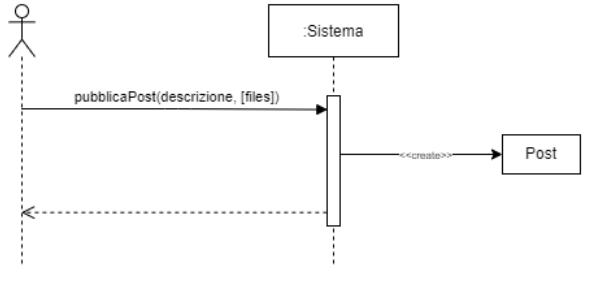


## PUBBLICA POST

PubblicaPost

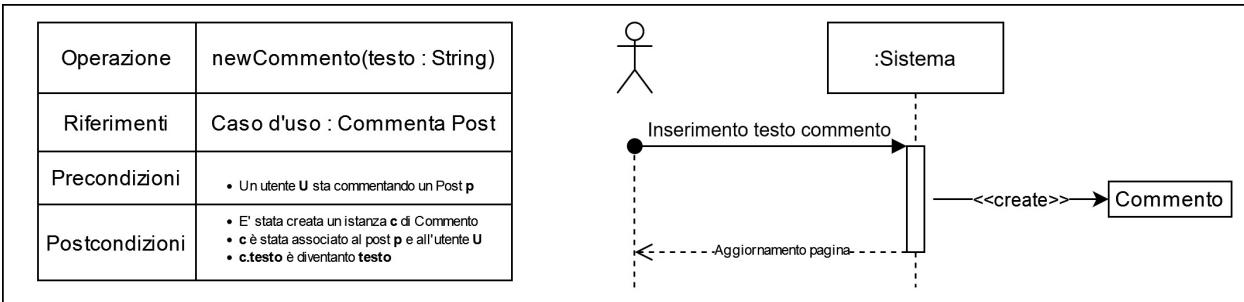
**PubblicaPost**

Operazione	pubblicaPost (descrizione: String, files: [File])
Riferimenti	Caso d'uso: PubblicaPost
Precondizioni	<ul style="list-style-type: none"> <li>Un utente <b>U</b> sta pubblicando un post</li> <li>È in corso una pubblicazione di un post</li> </ul>
Postcondizioni	<ul style="list-style-type: none"> <li>È stata creata un'istanza <b>p</b> di Post</li> <li>p è stato associato all'utente <b>U</b></li> <li>A <b>U</b> è stato associato <b>p</b></li> <li>p.descrizione è diventata <b>descrizione</b></li> <li>p.files è diventato <b>files</b></li> </ul>



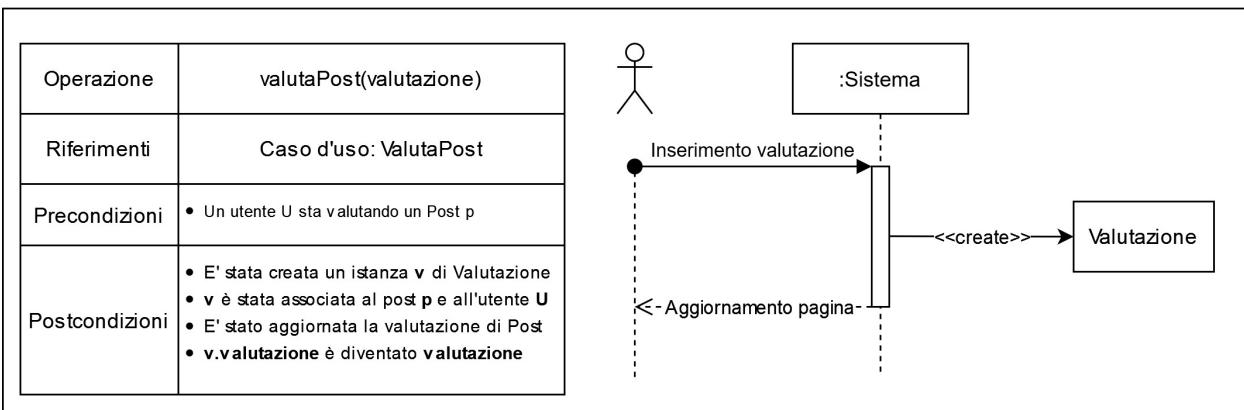
## COMMENTA POST

### CommentaPost



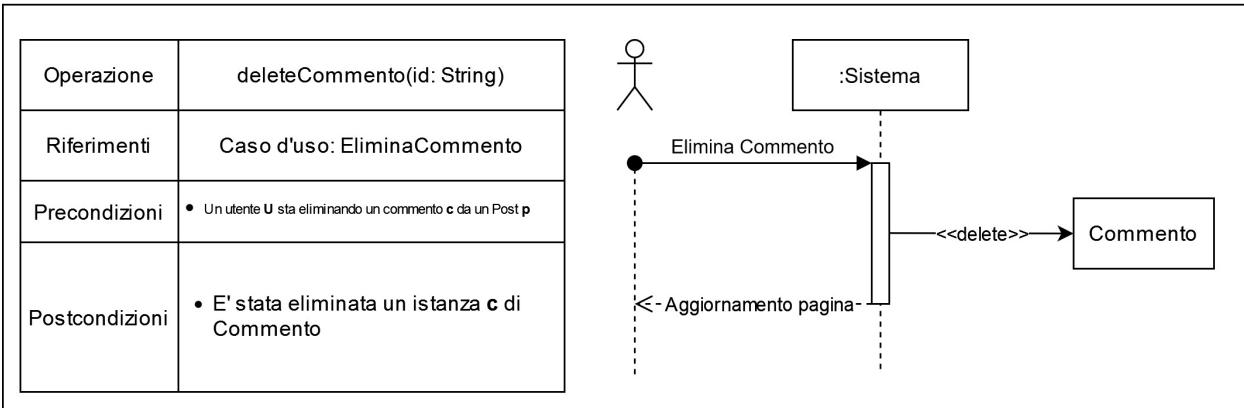
### VALUTA POST

ValutaPost



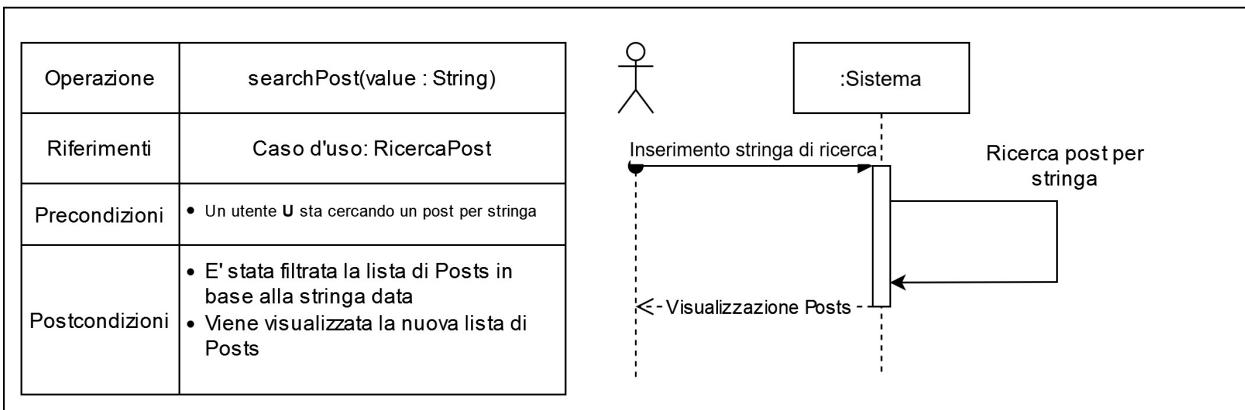
### ELIMINA COMMENTO

EliminaCommento



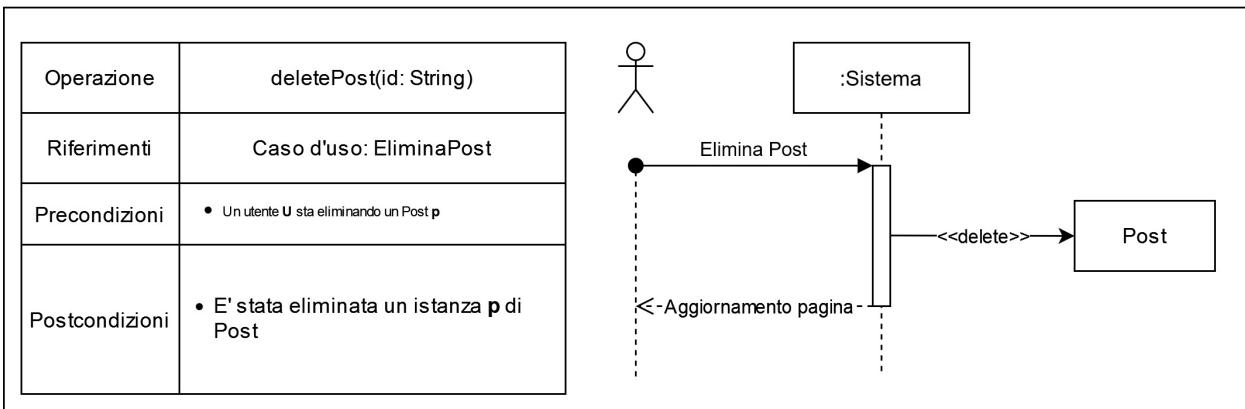
### RICERCA POST

### RicercaPost



### ELIMINA POST

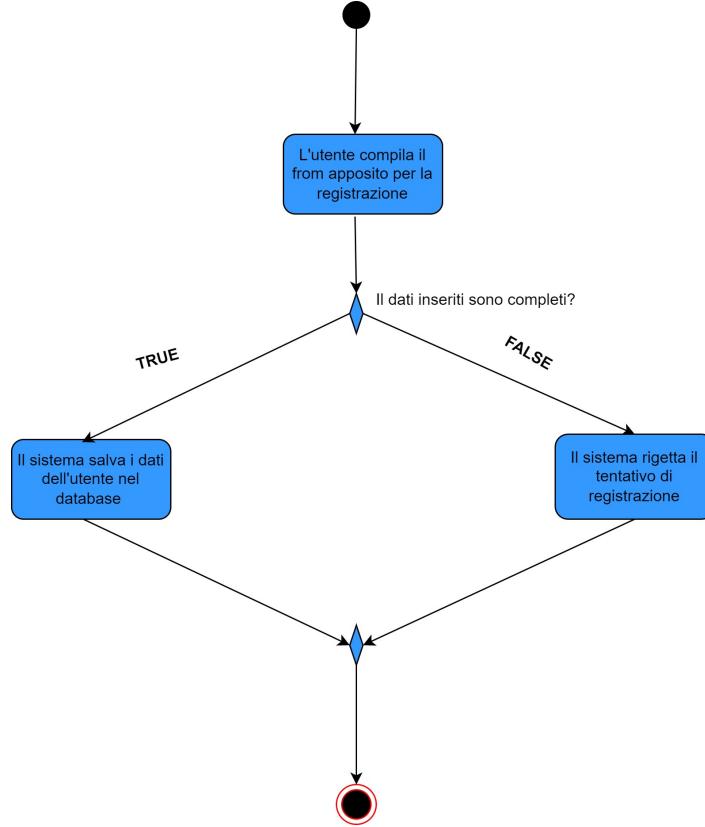
#### EliminaPost



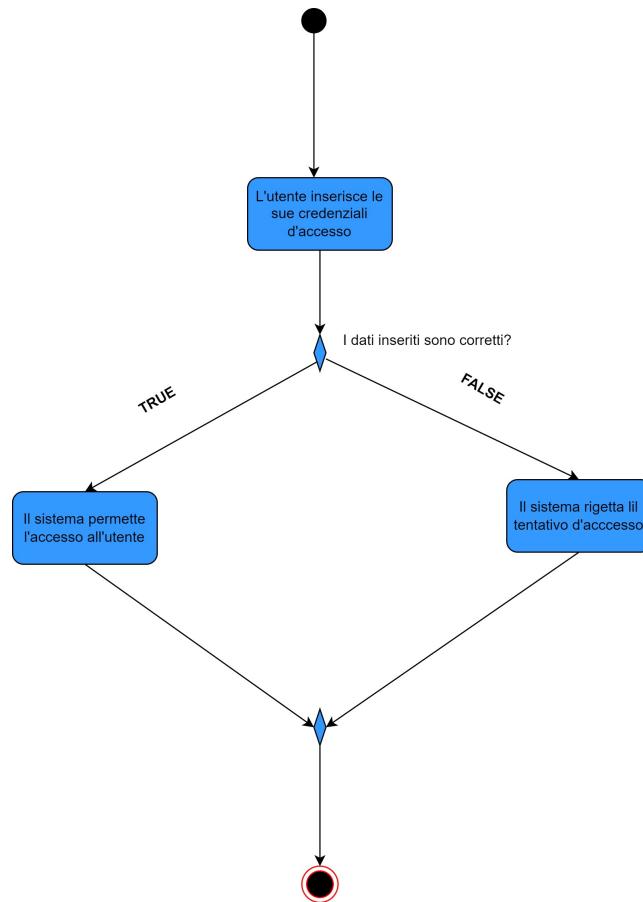
## 5.6 Activity Diagram

Gli activity diagram consentono di descrivere un processo attraverso un grafo orientato in cui i nodi rappresentano le attività e gli archi l'ordine in cui vengono eseguite le attività.

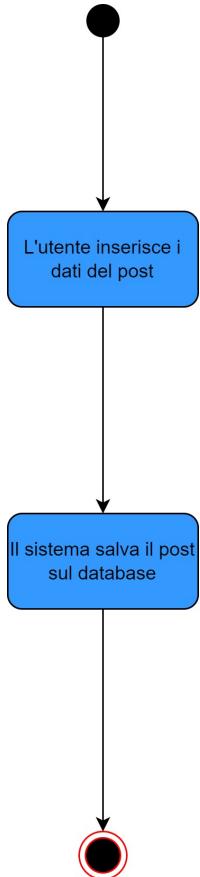
### Activity-Diagram : Registrazione



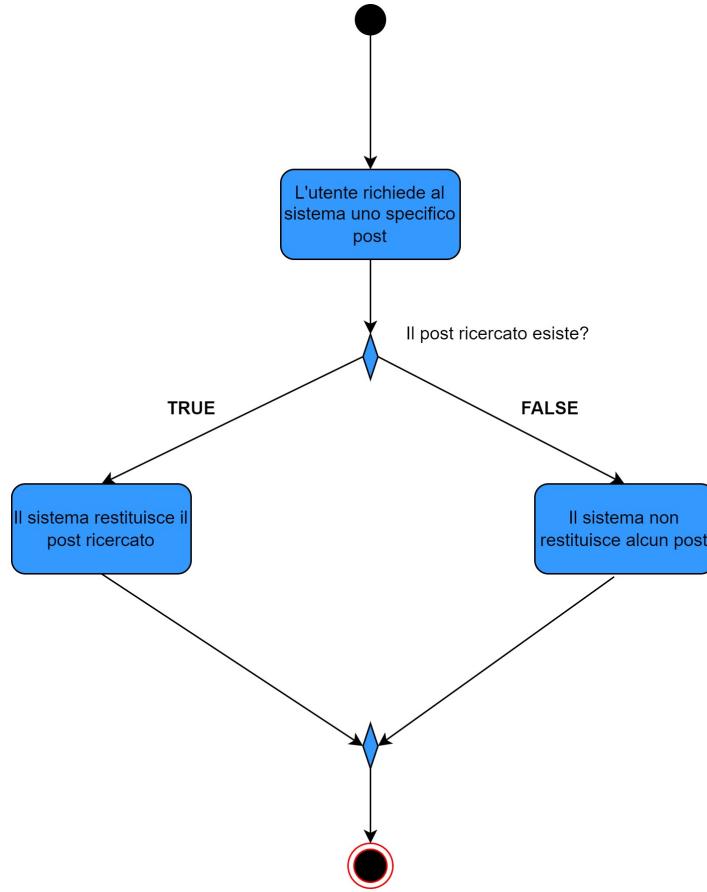
Activity-Diagram : Login



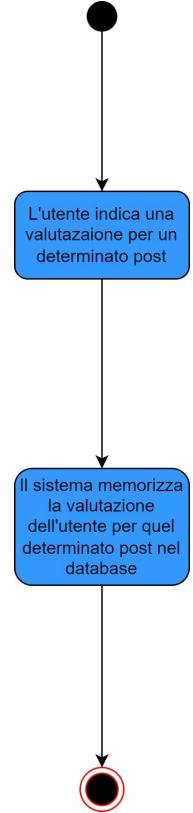
Activity-Diagram : PubblicaPost



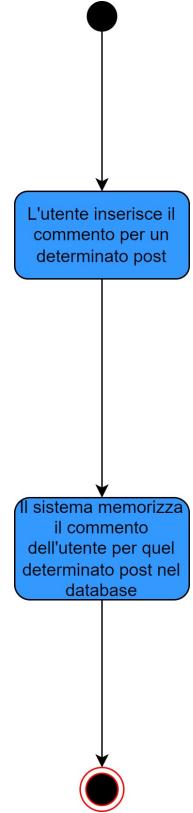
Activity-Diagram : RicercaPost



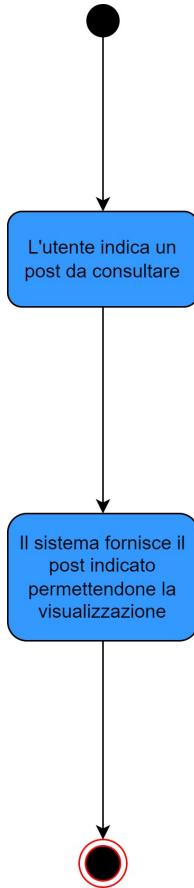
Activity-Diagram : ValutazionePost



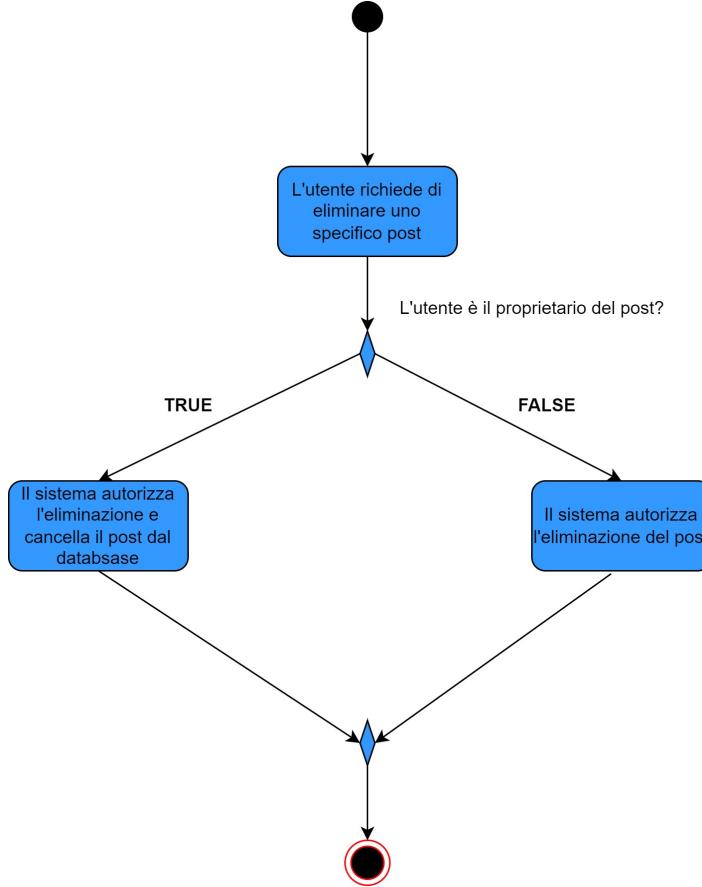
Activity-Diagram : CommentaPost



Activity-Diagram : ConsultaPost



Activity-Diagram : EliminaPost



## 6 Architettura software

All'interno di un sistema software è fondamentale avere una vista di come sarà l'architettura di quest'ultimo. In particolare un'architettura software è definita dai suoi componenti, dalle loro relazioni e dai principi che ne governano la progettazione e lo sviluppo.

Descrivere l'architettura software di un sistema significa individuare le parti costituenti ed illustrarne i rapporti inter-funzionali.

In questo caso si è avvalso per la descrizione di questo sistema software di uno stile architetturale di tipo **Client-Server-Database 3-tiers**; in particolare tale stile permette di ripartire l'elaborazione dei dati tra client e server in modo tale di evitare performance di basso livello ed inoltre permette l'interoperabilità tra sistemi, dando la possibilità a diversi componenti di lavorare assieme. E' stato inoltre scelto di separare fisicamente il Server dal database in modo tale da decentralizzare la persistenza dei dati su un unico nodo, ottenendo una reliability maggiore della persistenza dei dati fornita dall'architettura cloud **MongoDB Atlas, un cloud database**.

Da un punto di vista logico, il sistema software `e suddiviso in livelli, di seguito riportati :

- **Presentation:** si occupa di interfacciarsi direttamente con gli attori del sistema. Questo layer è composto dai seguenti componenti :
  - Login , Register, Home , Post e new Post che risiedono su lato Server e sono responsabili del funzionamento delle pagine web dell'applicazione.
- **Business Logic:** si occupa dell'implementazione della logica del sistema. E' composta dai seguenti componenti:
  - Service(lato client):

- *appState.service*: modulo responsabile della memorizzazione delle informazioni dell'utente loggato in una determinata sessione sull'interfaccia grafica.
- *utente.service* : modulo che si occupa di inoltrare i dati dell'utente al nodo server tramite richieste HTTP.
- *post.service* : modulo che si occupa di inoltrare i dati riguardanti un post al nodo server tramite richieste HTTP.
- *valutazione.service*: modulo che si occupa di inoltrare le valutazioni riguardanti un post al nodo server tramite richieste HTTP.
- *commento.service:modulo*: che si occupa di inoltrare i commenti associati ad un post al nodo server tramite richieste HTTP.
- *segnalazione.service*: modulo che si occupa di inoltrare le segnalazioni riguardanti un post al nodo server tramite richieste HTTP.

I sopracitati moduli posso essere visualizzati nella figura 6.3.1 *Diagrammi architetturali di dettaglio lato client*.

Service (lato server) :

- *UtenteService* : modulo responsabile della gestione dei dati dell'utente all'interno della piattaforma.
- *PostService*: modulo responsabile della gestione dei post degli utenti all'interno della piattaforma.
- *CommentoService*: modulo responsabile della gestione dei commenti associati ai post all'interno della piattaforma
- *ValutazioneService*: modulo responsabile della gestione delle valutazioni associate ai post all'interno della piattaforma
- *FileService* : modulo responsabile della gestione dei file condivisi all'interno dei post sulla piattaforma
- *SegnalazioneService*: modulo responsabile delle segnalazioni sui post o sui commenti presenti all'interno della piattaforma.

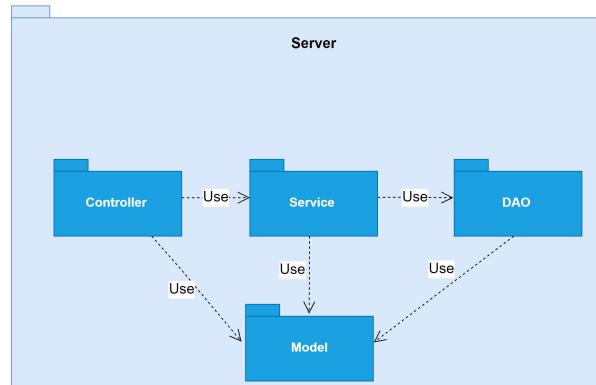
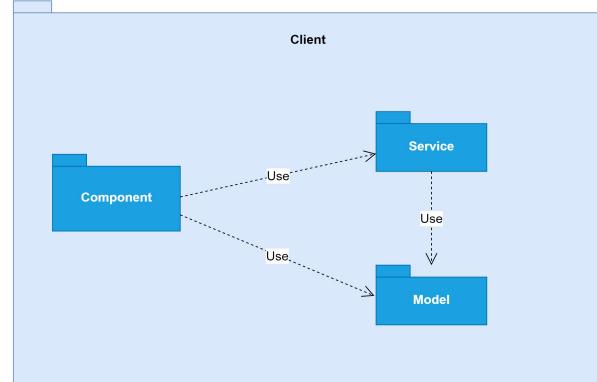
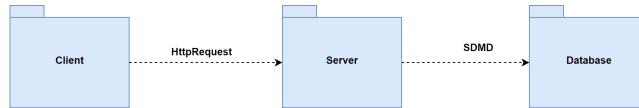
I sopracitati moduli posso essere visualizzati nella figura 6.3.12 *Diagrammi architetturali di dettaglio lato server*.

- **Data Access:** si occupa dell'interazione col database.

- DAO: insieme di moduli, uno per ogni entità del sistema, che interagiscono direttamente col database. Più nello specifico è stata separata l'interfaccia dall'implementazione per ogni DAO in modo tale da essere svincolati da eventuali migrazioni su altre base dati.

I moduli DAO posso essere apprezzati nella figura 6.3.12 *Diagrammi architetturali di dettaglio lato server*.

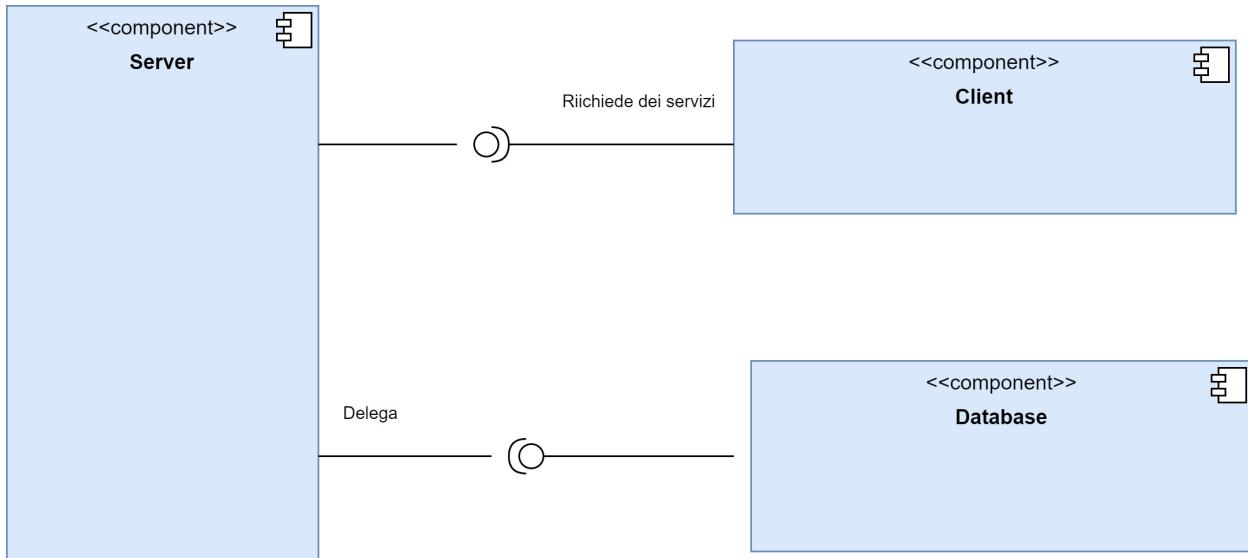
## 6.1 Viste architetturali di alto livello



Nella prima figura per **SDMD (Spring Data MongoDB)** si intendono tutte quelle API che il framework Spring mette a disposizione per comunicare con il database MongoDB.

## 6.2 Component diagram

Per avere una vista della struttura interna del software vari componenti è utile stendere un diagramma dei componenti che modella le interazioni tra i vari componenti del sistema stesso.

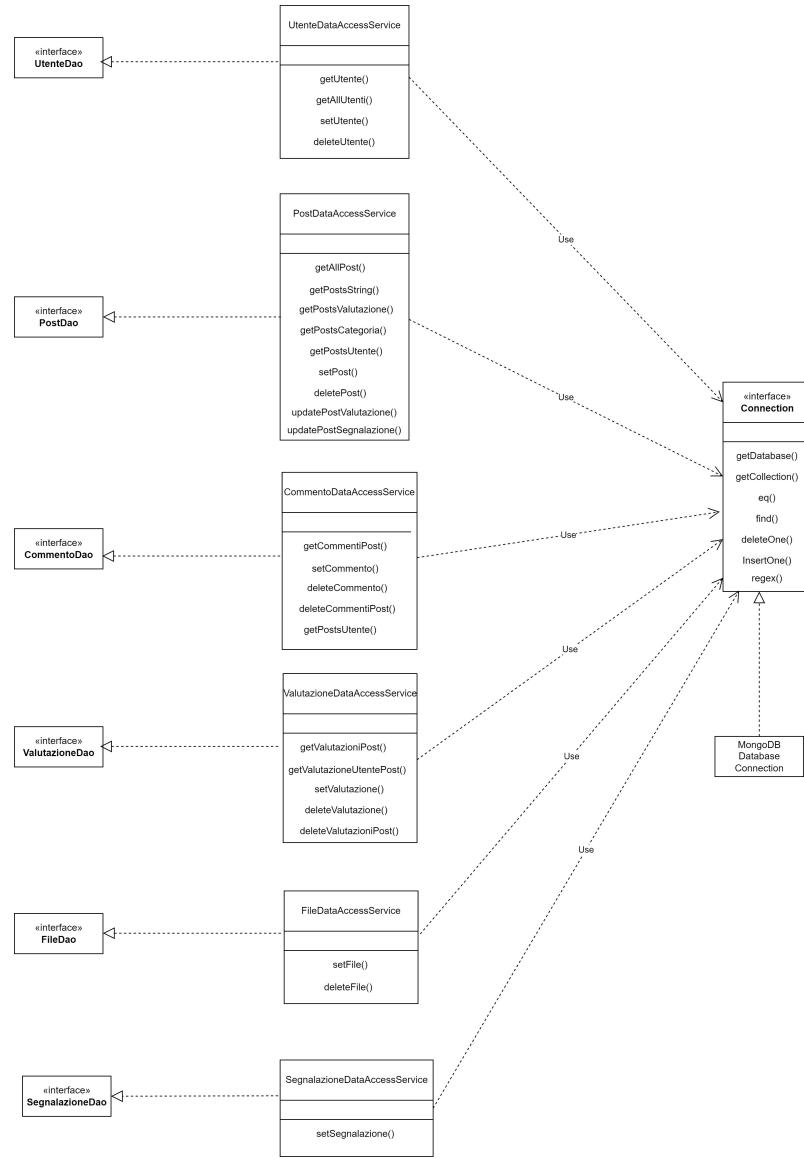


**6.2.1 Diagramma dei componenti dettagliato**



### 6.2.2 Diagrammi delle interfacce Server-Database

Più nello specifico andremo a mostrare quali sono le interfacce di comunicazione tra server e database, i metodi messi a disposizione dal database per l'accesso ai dati, i quelli che precedentemente abbiamo chiamato SDMD.



Nel paragrafo successivo verrà dettagliata l'architettura del Server ed è così che questo component diagram potrà arricchire la comprensione di quest'ultimo in relazione al Database.

## 6.3 Descrizione dell'architettura di dettaglio

### 6.3.1 Diagrammi architetturali di dettaglio lato client



Il Client è sviluppato usando il framework **Angular**.

Il core della logica di Angular sono i **componenti**, i quali rappresentano le pagine web della web application. Ogni componente è encapsulato in dei package ed ogni package contiene tre moduli che sono responsabili dell'implementazione della struttura e del funzionamento di ogni pagina web. Più nello specifico i tre moduli sono:

**1)component.html** : è un modulo html che permette la scrittura in linguaggio HTML della struttura della pagina web che consiste nella sua formattazione e impaginazione.

**2)component.ccs** : è un modulo ccs per la stilizzazione della pagina web.

**3)component.ts** : è un modulo TypeScript in cui sono organizzati le variabili e i metodi che permettono l'interazione utente con la pagina web.

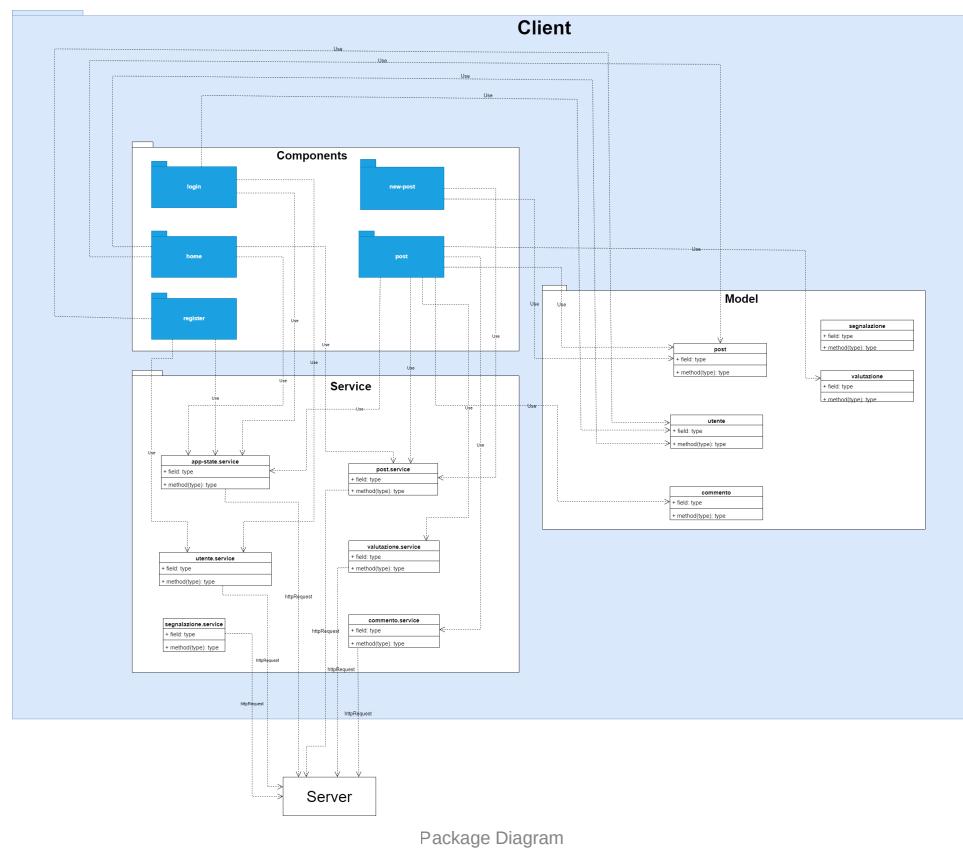
In aggiunta l'intera applicazione è considerata un componente e segue la stessa struttura sopracitata.

In conclusione è giusto citare altri due moduli di fondamentale importanza:

-**app-routing.module.ts** : che si occupa di associare ad ogni componente un path specifico che sarà inserito nell'address bar.

-**app.module.ts** : che è responsabile dell'importazioni di componenti esterni da cui è possibile attingere moduli di utilità per l'utilizzo all'interno dell'applicazione.

**N.B.** In questa applicazione, i component.ccs che di default sono creati da Angular sono stati sostituiti con dei component.sccs i quali contengono più features rispetto a quelli css.



Come è possibile apprezzare da questo diagramma nel caso specifico della nostra web application i componenti che la compongono sono cinque:

**1)login Component.**

**2)register Component.**

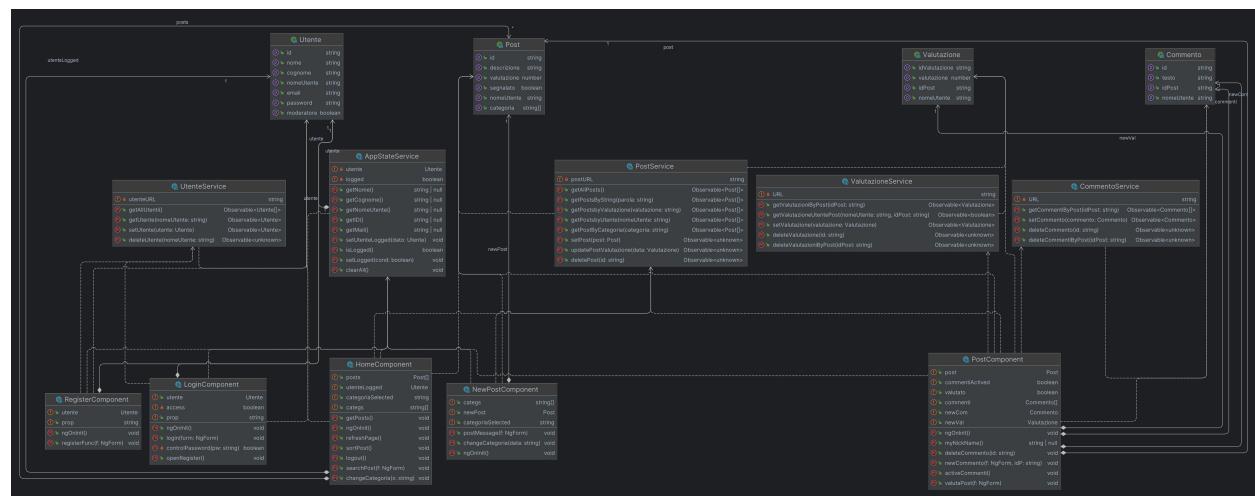
3)home Component.

4)post Component.

5)new-post Component.

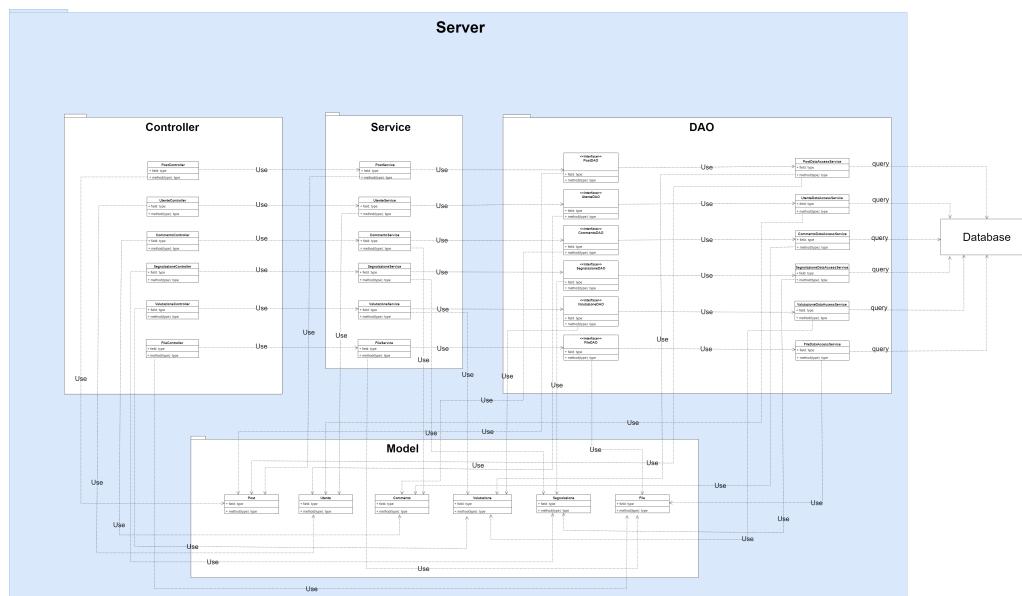
Per una questione di complessità di scrittura di codice e di organizzazione di sviluppo software è stato deciso di separare in componenti diversi Home, Post , New Post anche se appartengono alla stessa pagina web.

Nel package Service è incapsulata la logica di funzionamento delle pagine web . Ogni pagina web utilizza i servizi presenti all'interno del service specifico per poter richiedere al Server le informazioni necessarie per completare un determinato caso d'uso. I service , uno per ogni entità nel modello di domino ,esclusa l'entità file che non è stata implementata, non fanno altro che mappare i metodi chiamati dall'utente tramite l'interfaccia grafica(pagina web) in richieste HTTP che saranno inoltrate al Server. C'è da segnalare un service anomalo nel comportamento, ovvero **app-state.service** che si occupa di salvare i dati utente al momento del login, per l'intera sessione, in un dizionario.



Class Diagram

### 6.3.2 Diagrammi architetturali di dettaglio lato server

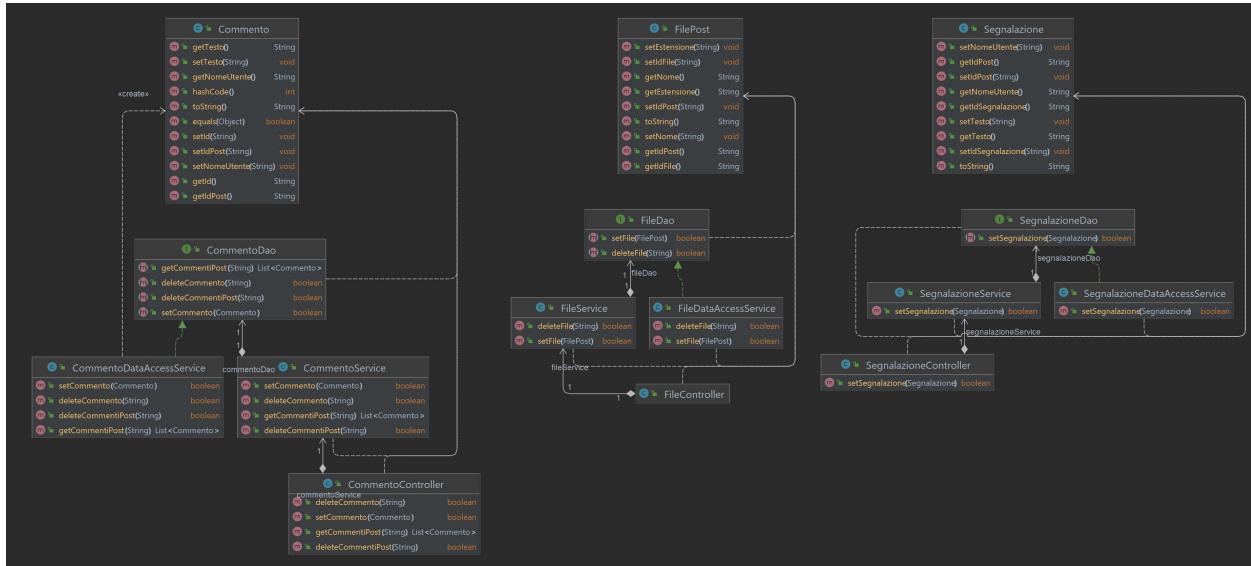


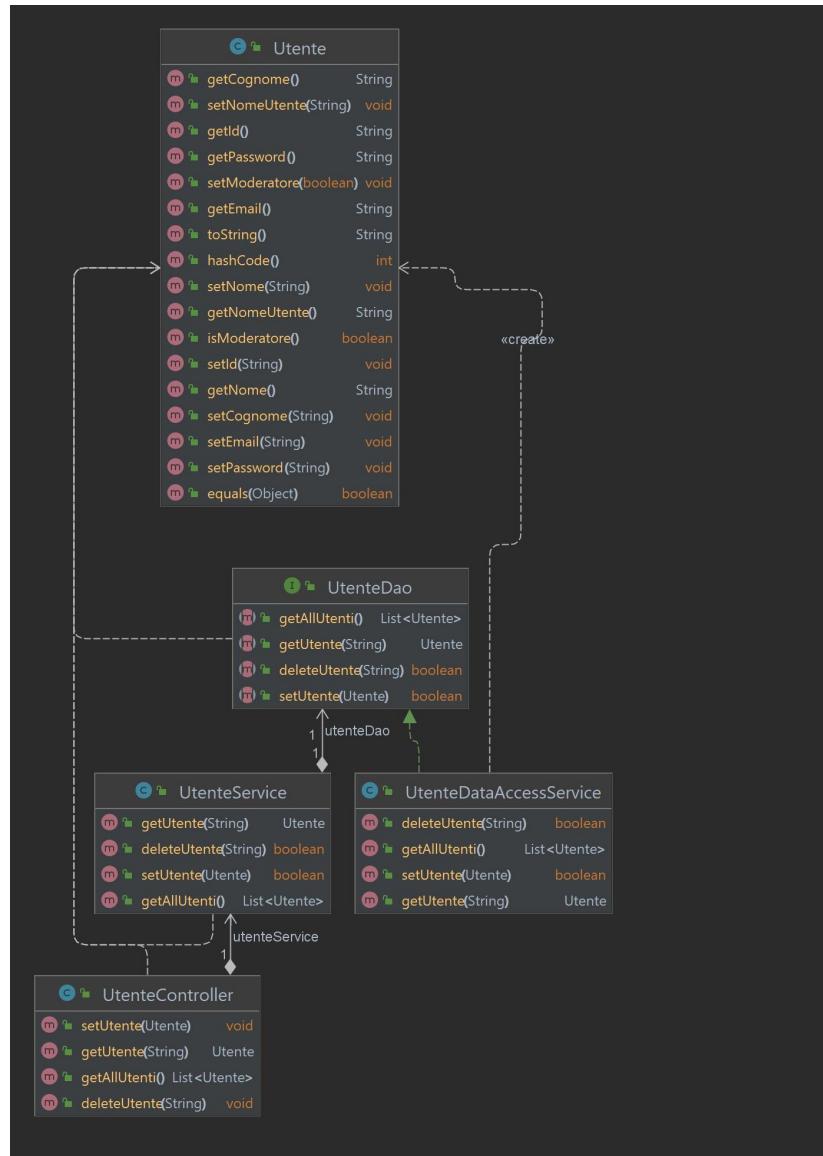
Package Diagram

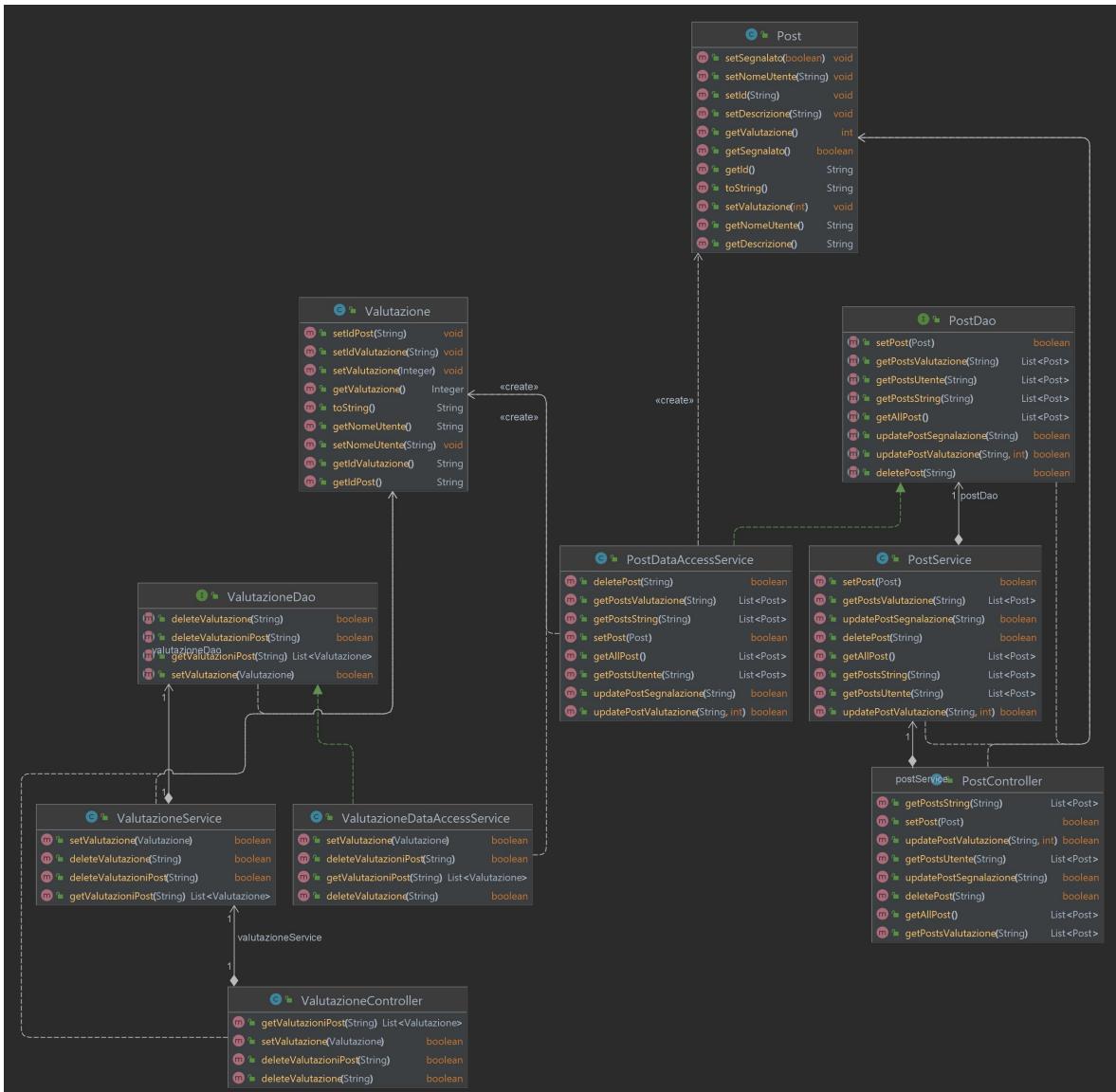


Il Server è stato implementato usando il framework SpringBoot. Il server è costituito da 4 package:

- **Controller**: il quale mappa gli URL per le richieste HTTP che vengono inoltrate al server.
- **Service**: collega il livello del Controller con il DAO, e permette di richiamare le funzionalità delle classi DAO.
- **DAO**: qui avviene il collegamento con il database e le effettive richieste a quest'ultimo per le azioni CRUD (GET, POST, UPDATE, DELETE).
- **Model**: le classi che definiscono la struttura dei diversi oggetti utilizzati all'interno del Server (es. Utente, Post, Commento, . . . etc.).





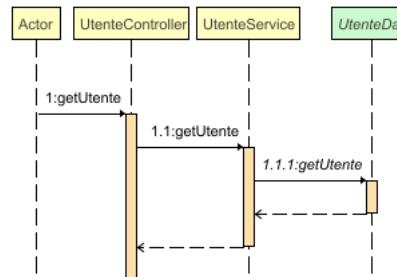
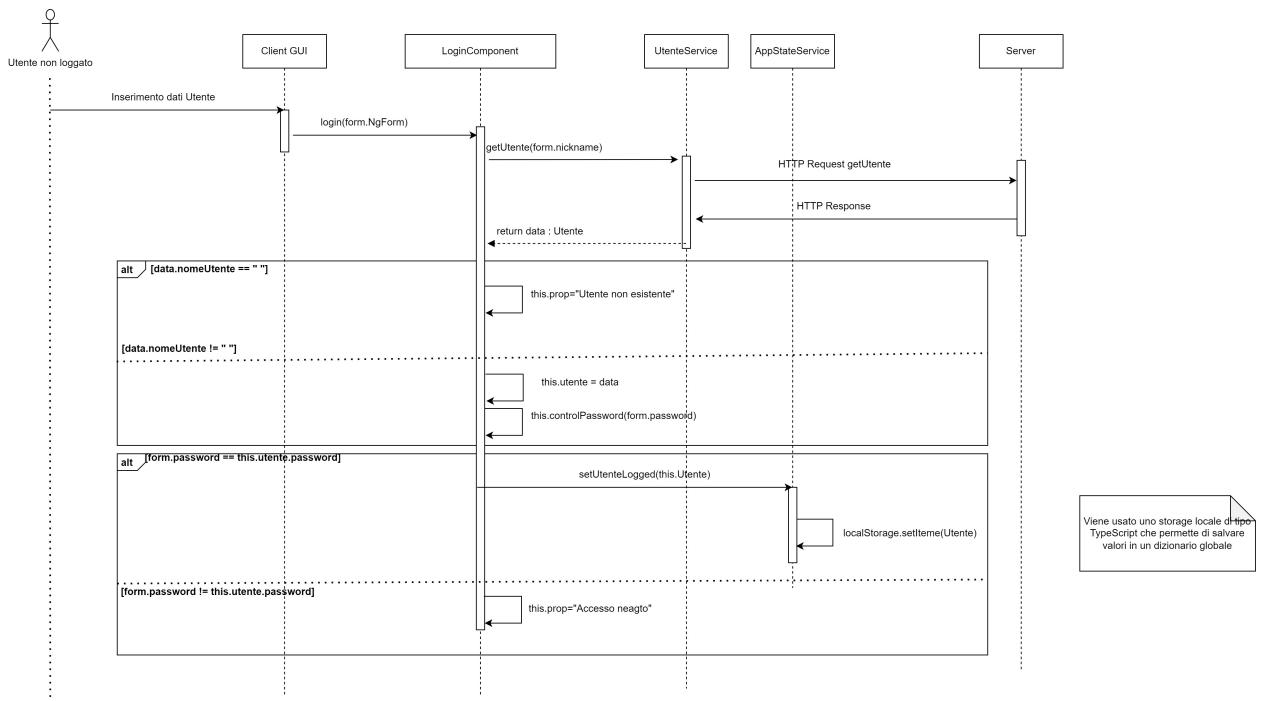


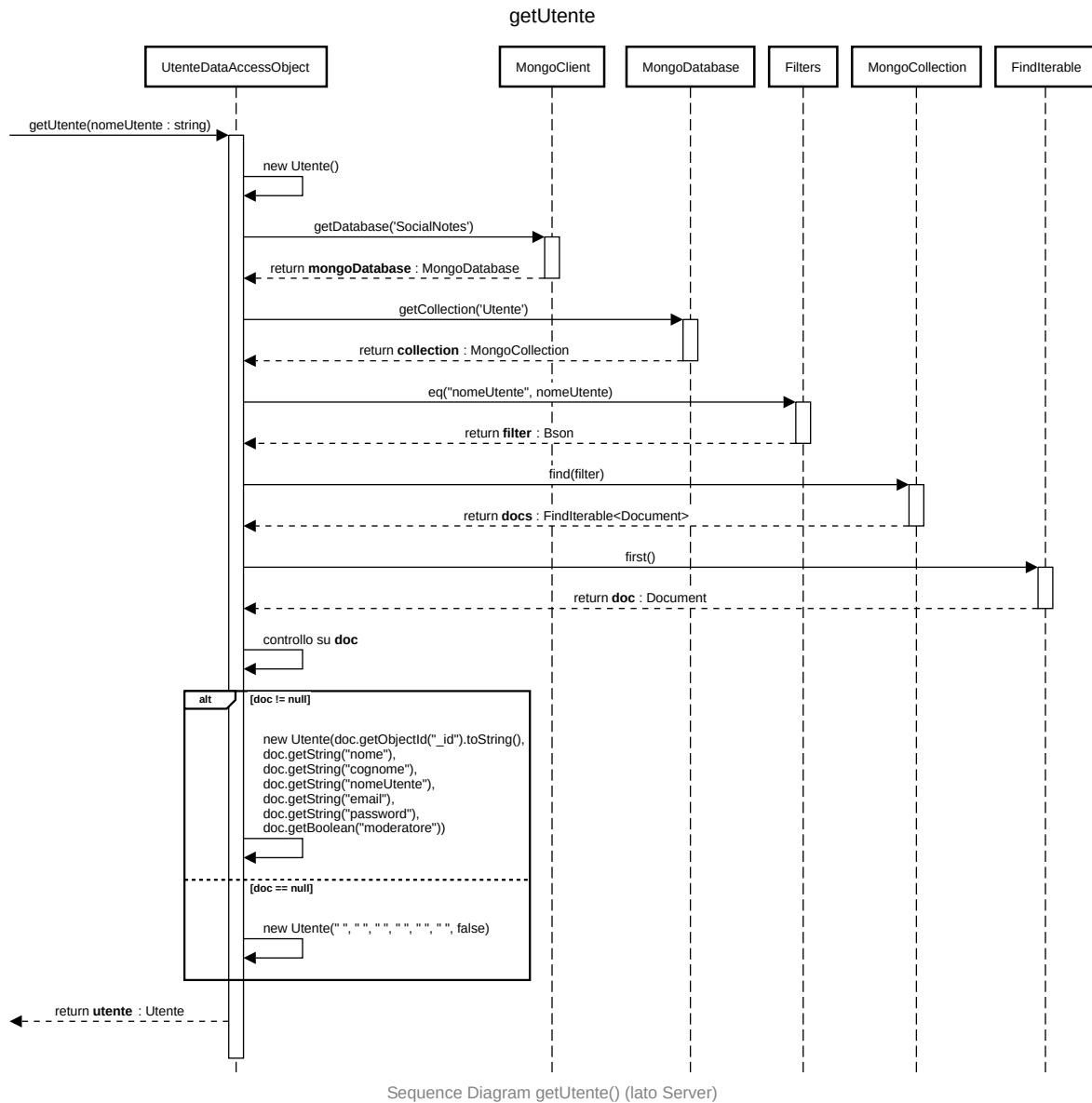
### 6.3.3 Sequence Diagram di dettaglio



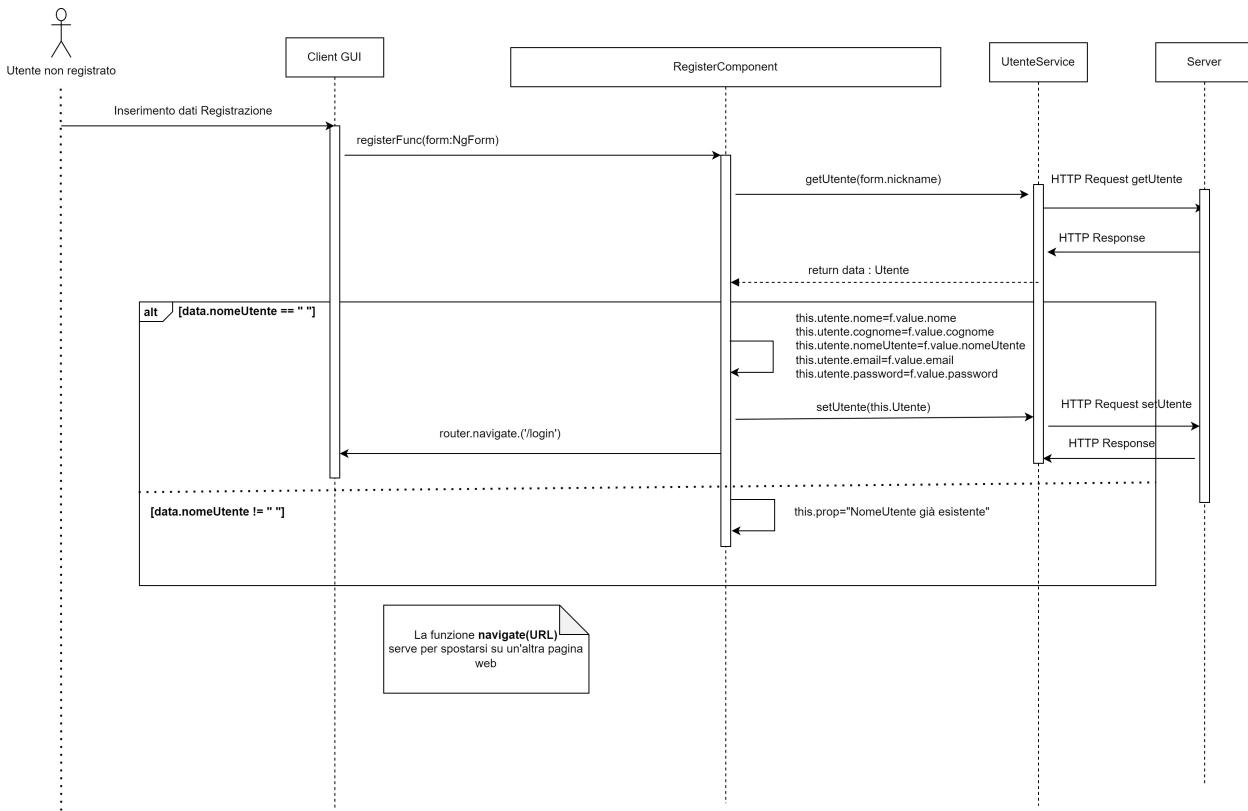
Di seguito sono riportati i Sequence Diagram dei casi d'uso che sono stati implementati nel sistema.  
Per ogni caso d'uso è riportato il Diagramma sia per il Client che per il Server.

#### Sequence diagram: [LOGIN\(Client\)](#)

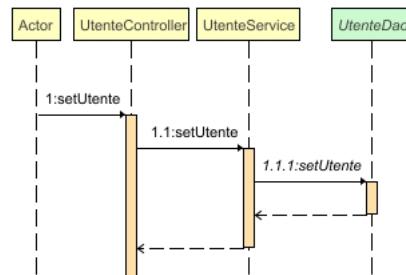




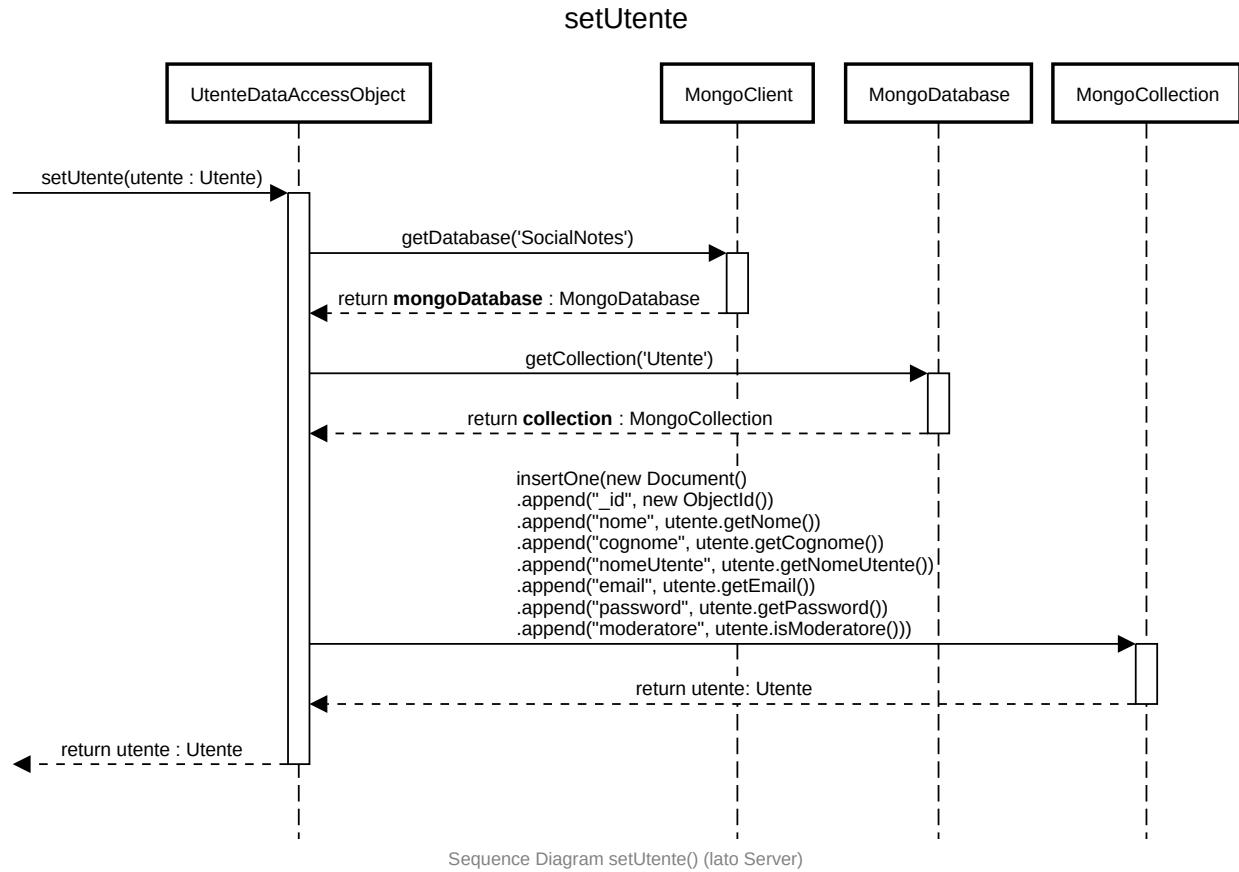
Sequence diagram: REGISTRA(Client)



**Sequence Diagram :Registra(Server)**

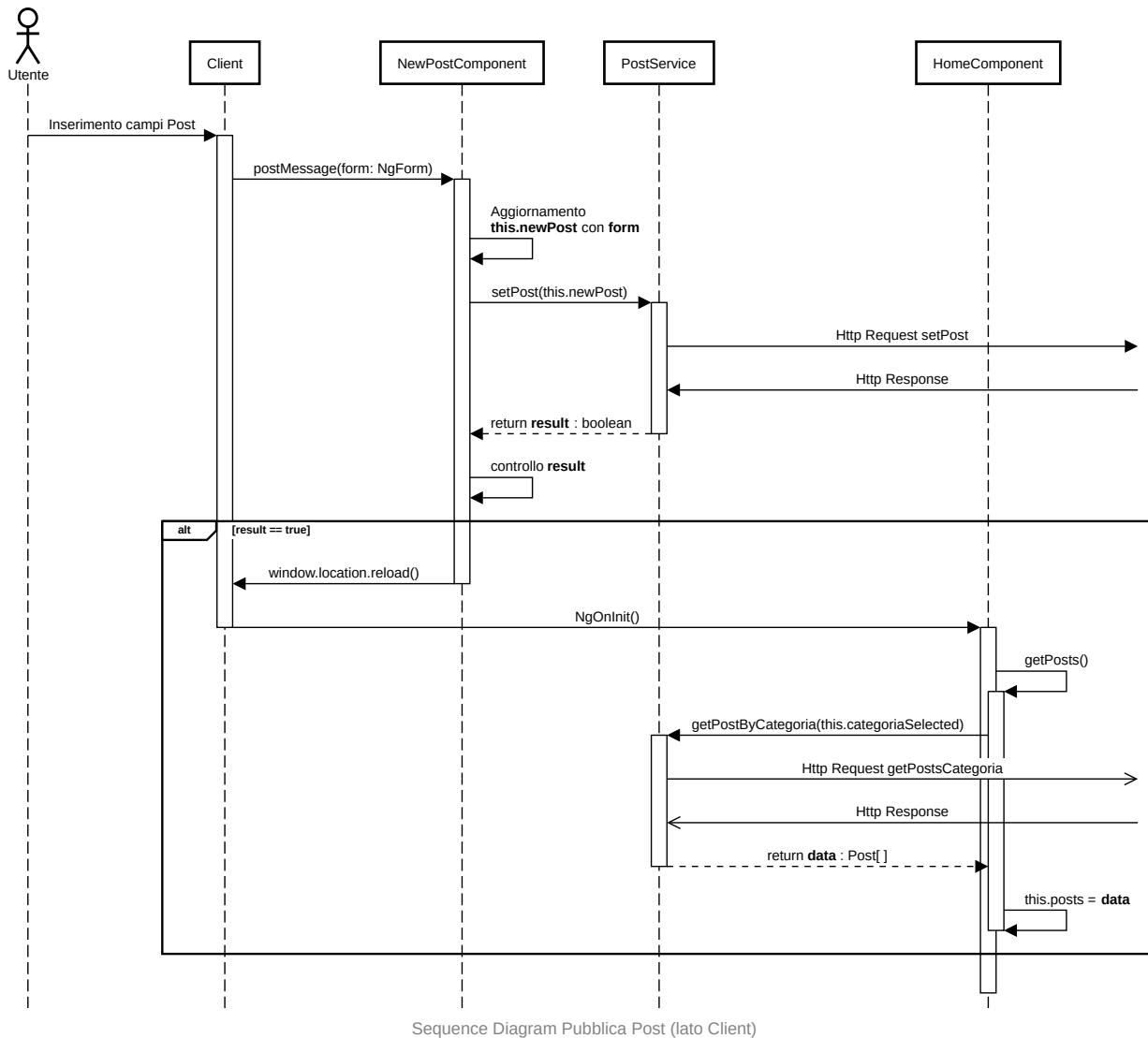


Sequence Diagram `setUtente()` (lato Server)



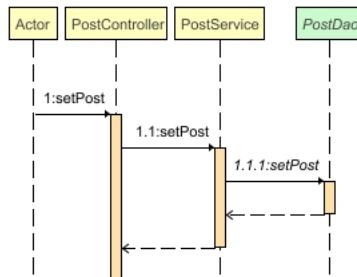
Sequence Diagram: PUBBLICA POST(Client)

### Pubblica Post

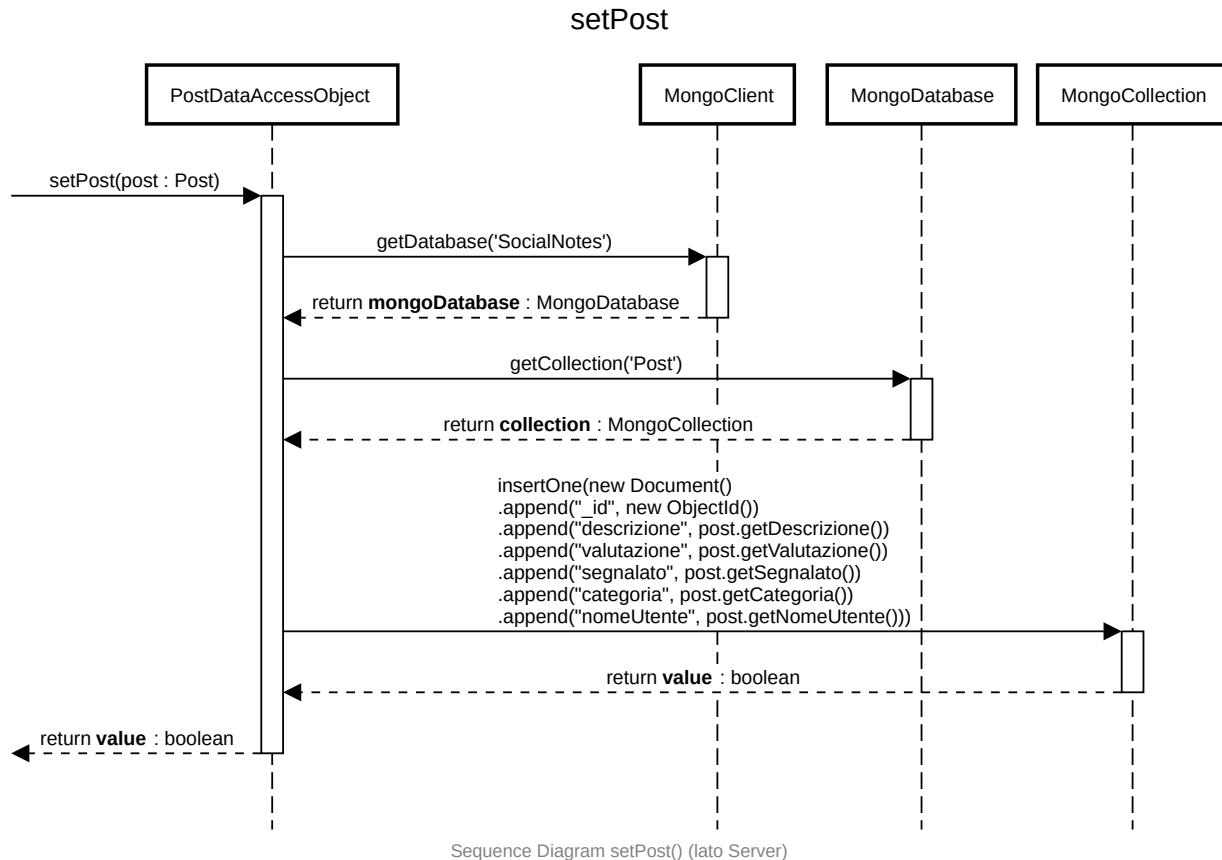


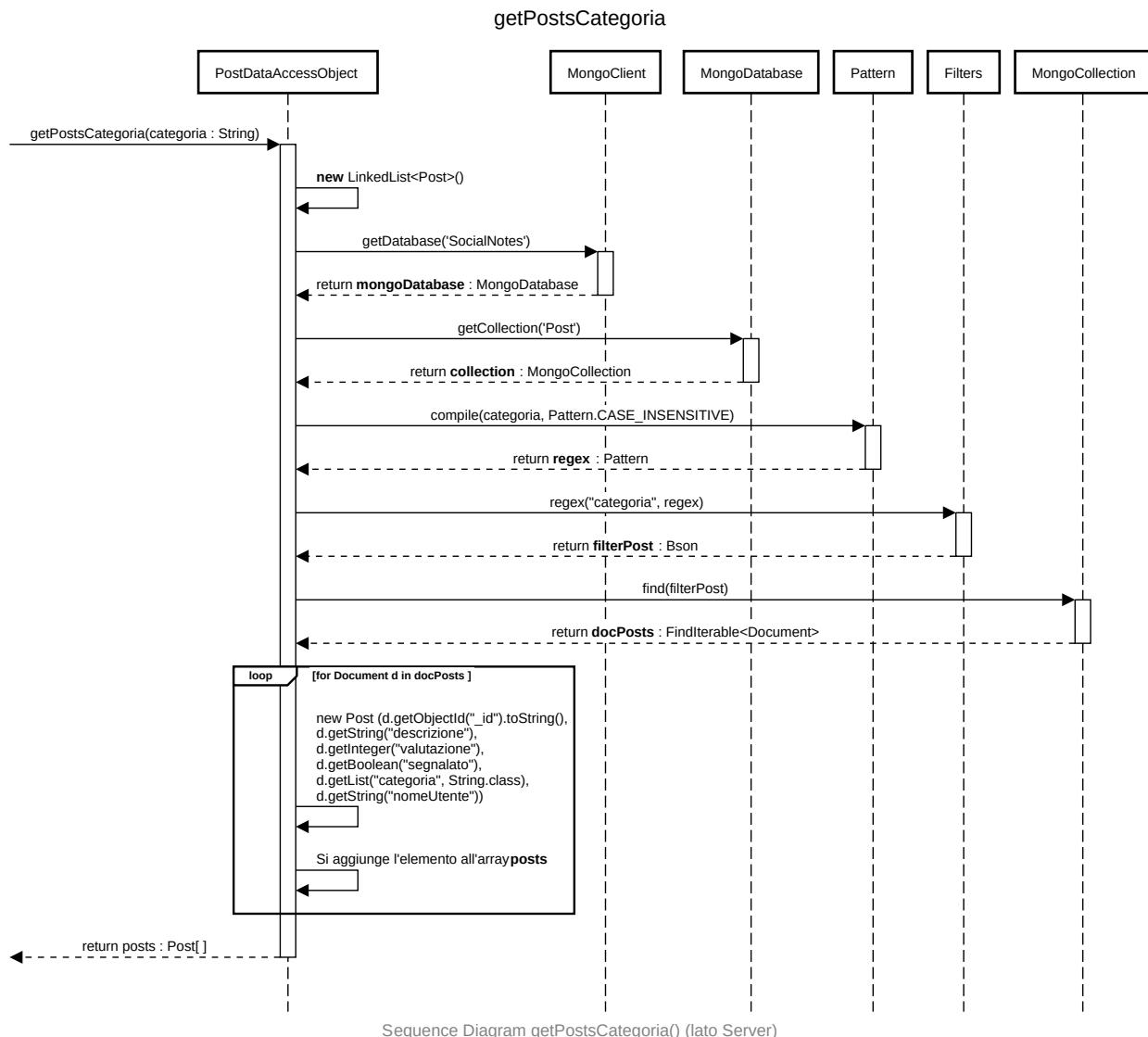
Sequence Diagram Pubblica Post (lato Client)

### Sequence Diagram: PubblicaPost(Server)

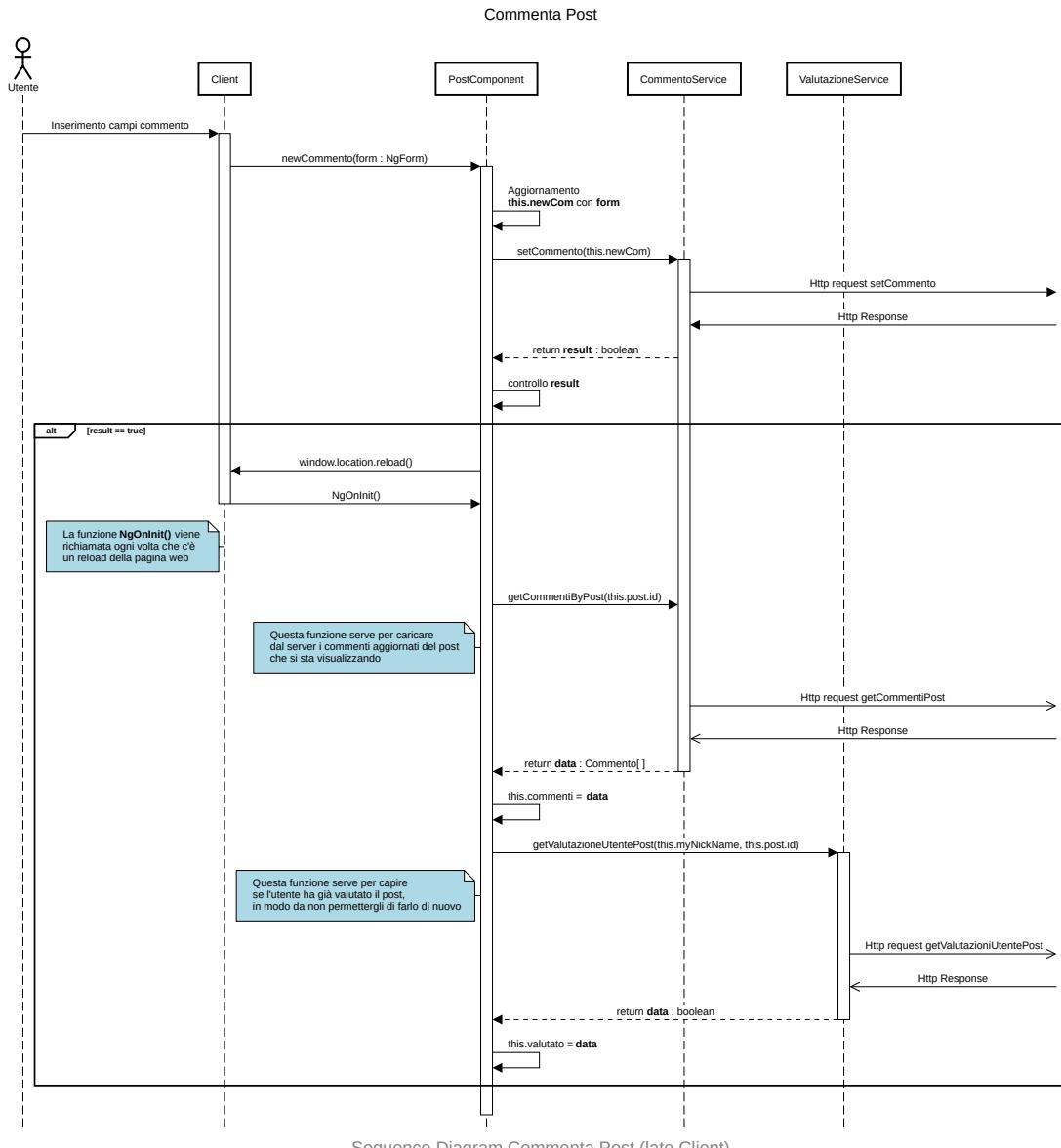


Sequence Diagram setPost() (lato Server)

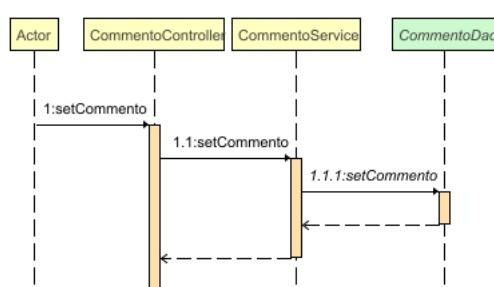




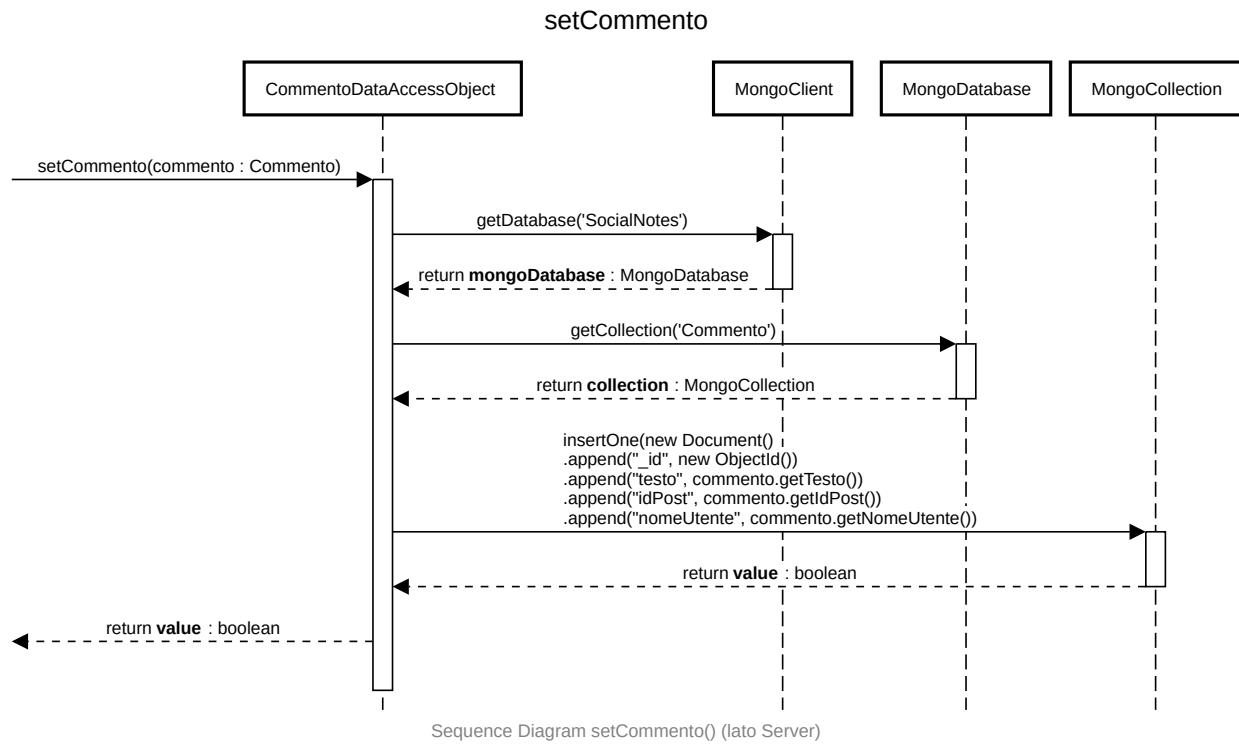
Sequence Diagram: **COMMENTA POST(Client)**



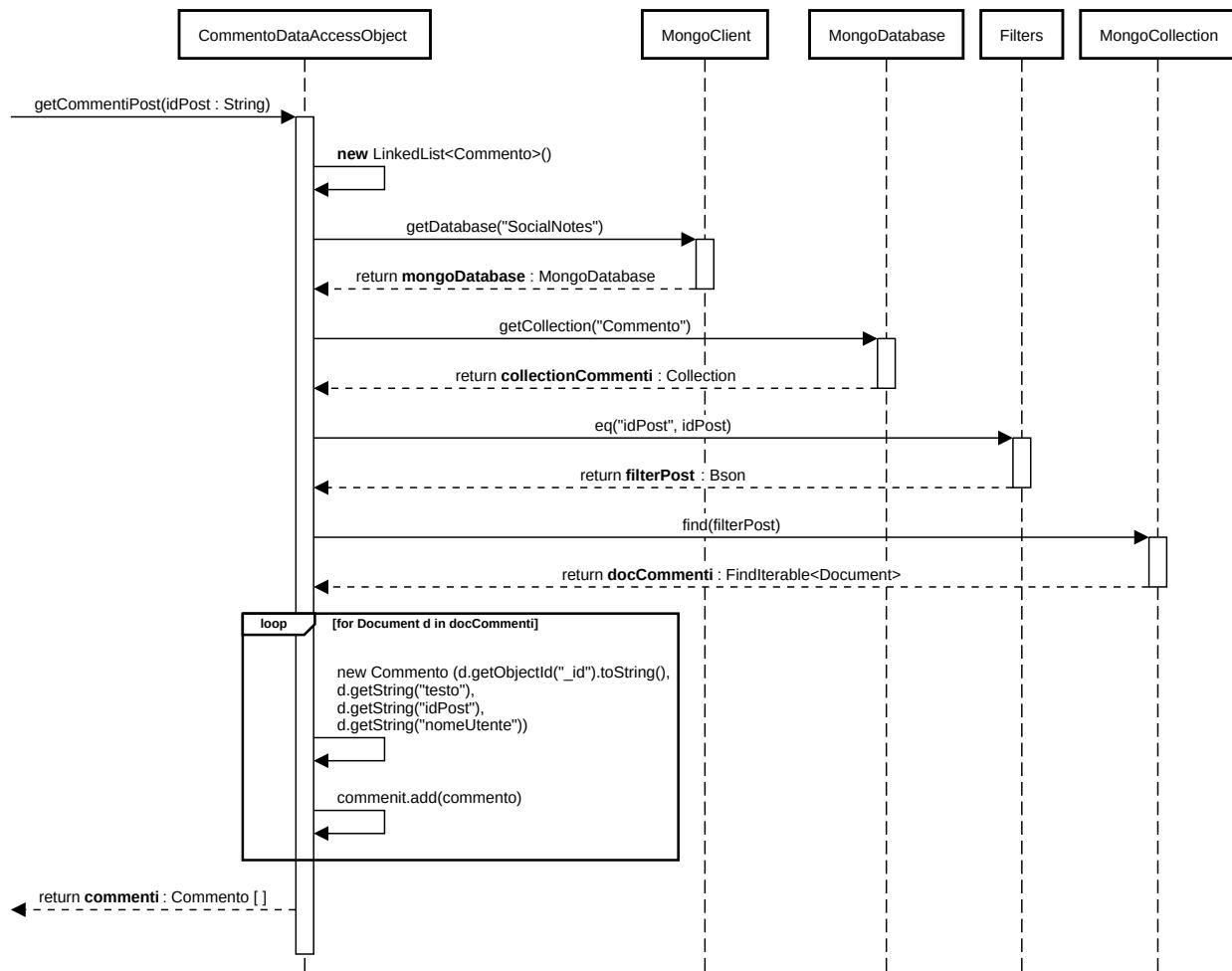
#### Sequence Diagram: CommentaPost(Server)



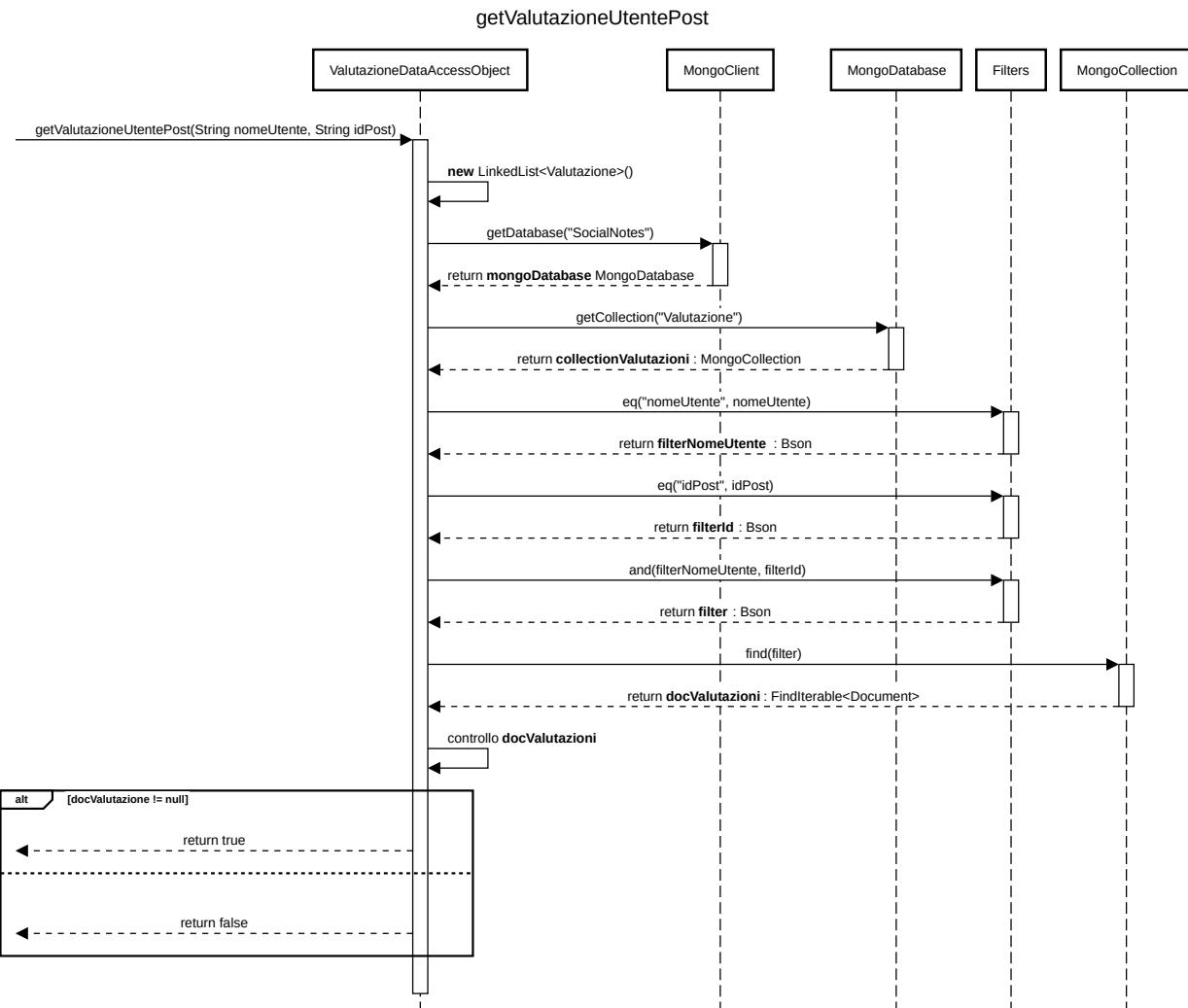
Sequence Diagram setCommento() (lato Server)



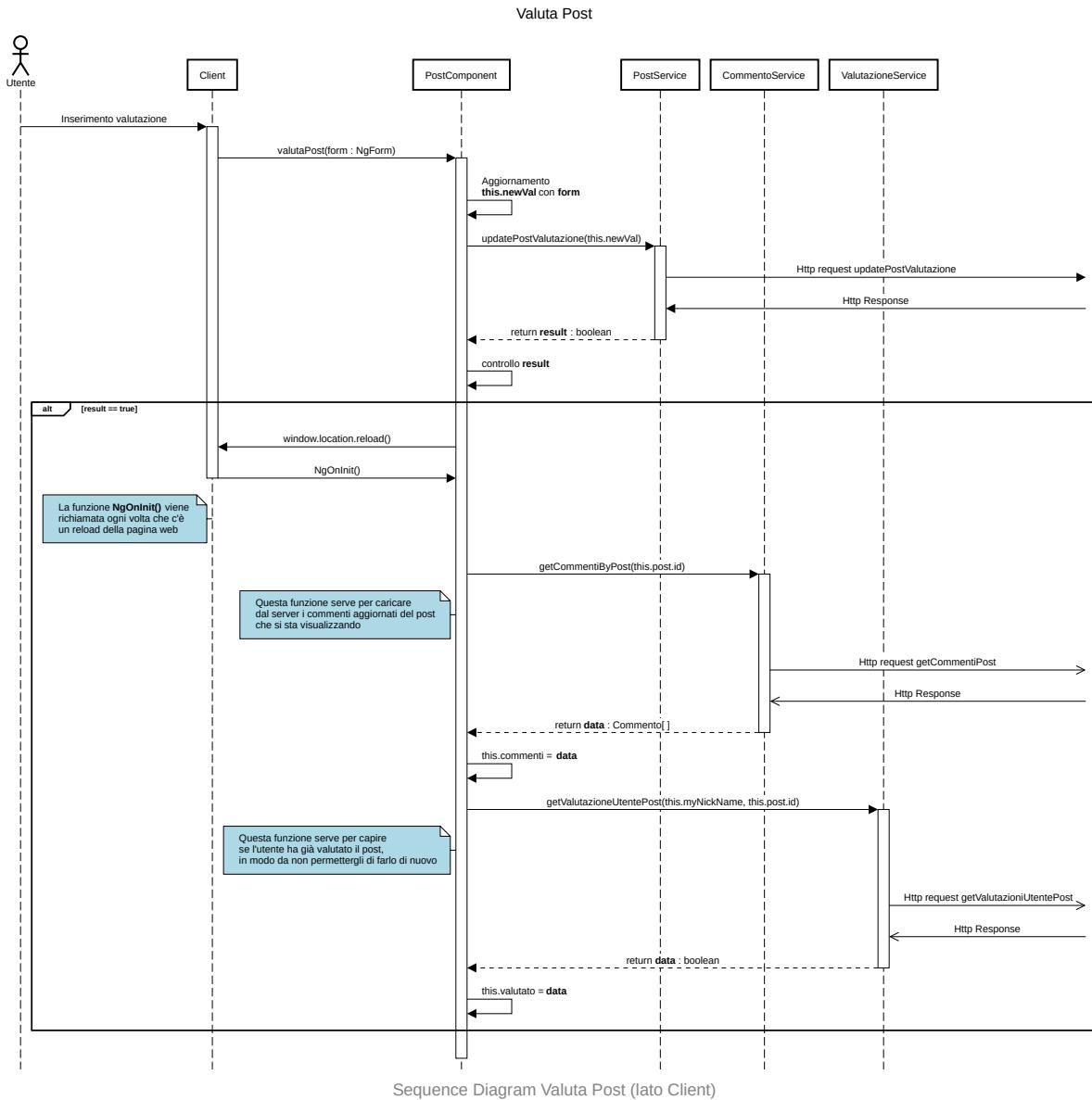
### getCommentiPost



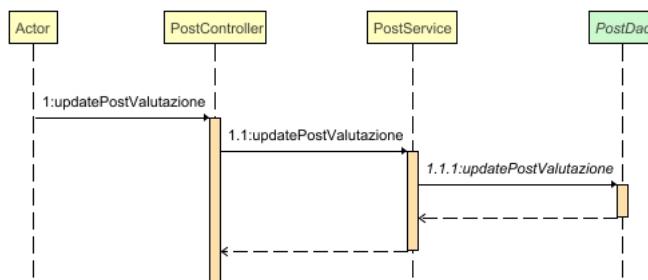
Sequence Diagram getCommentiPost() (lato Server)



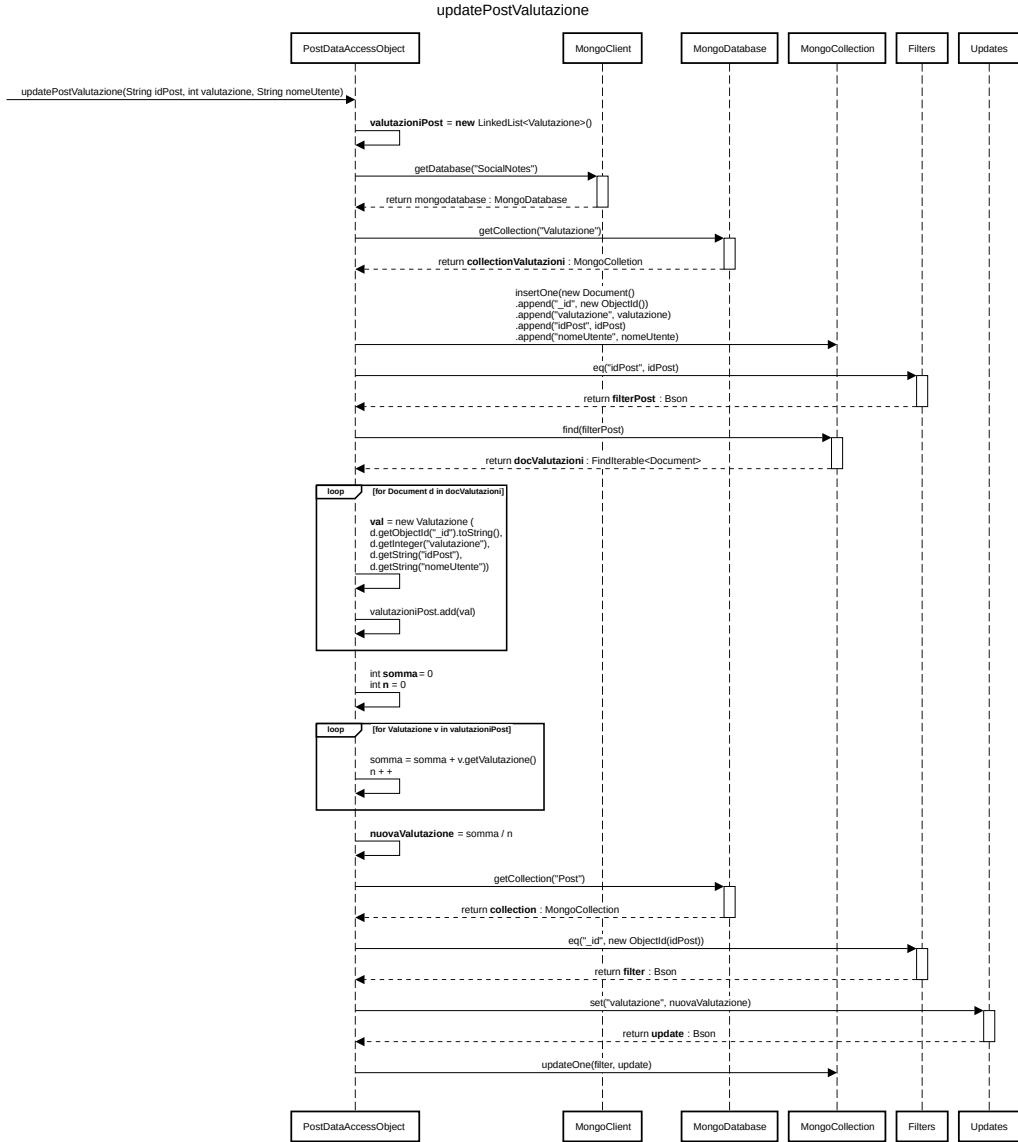
Sequence Diagram: VALUTA POST(Client)

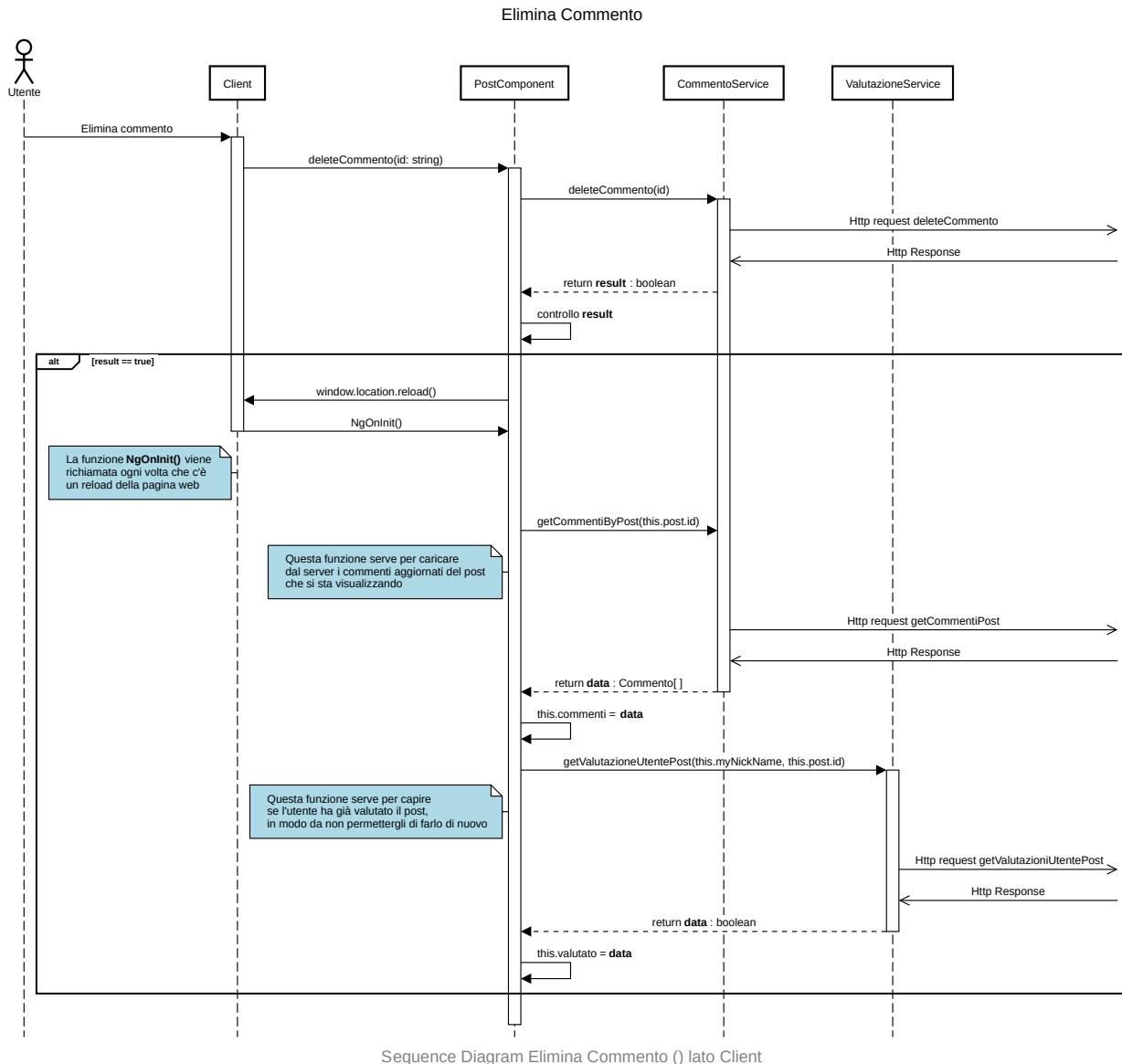


#### Sequence Diagram: ValutaPost(Server)

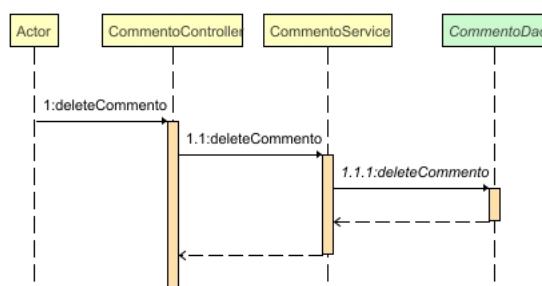


Sequence Diagram updatePostValutazione() (lato Server)

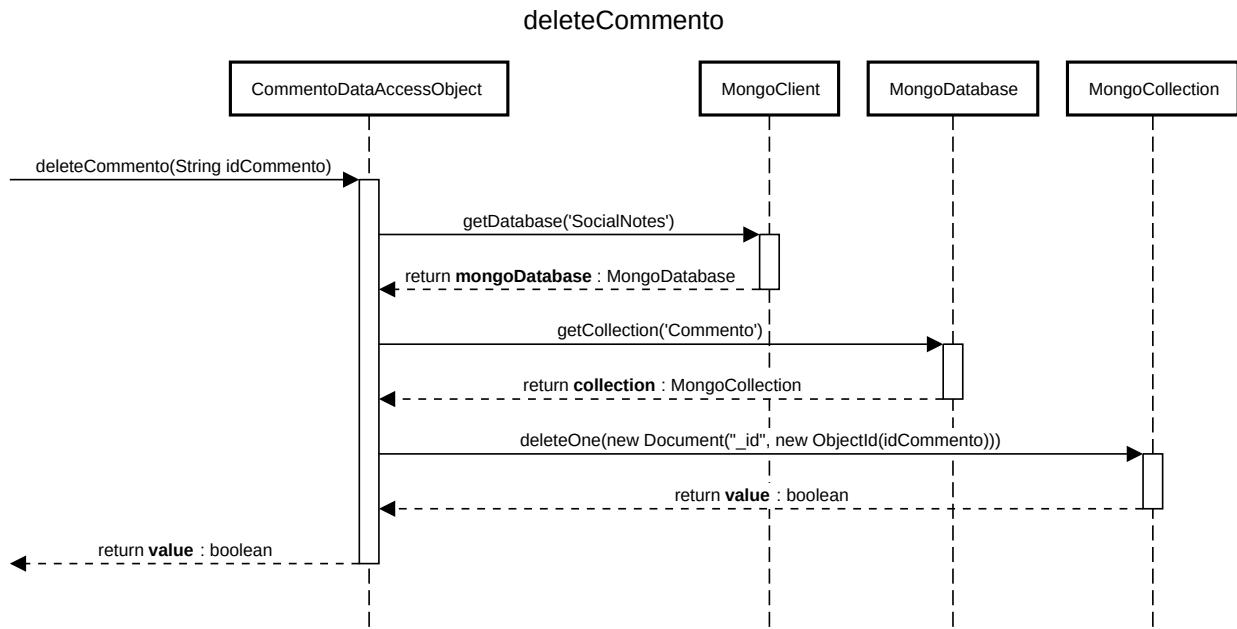




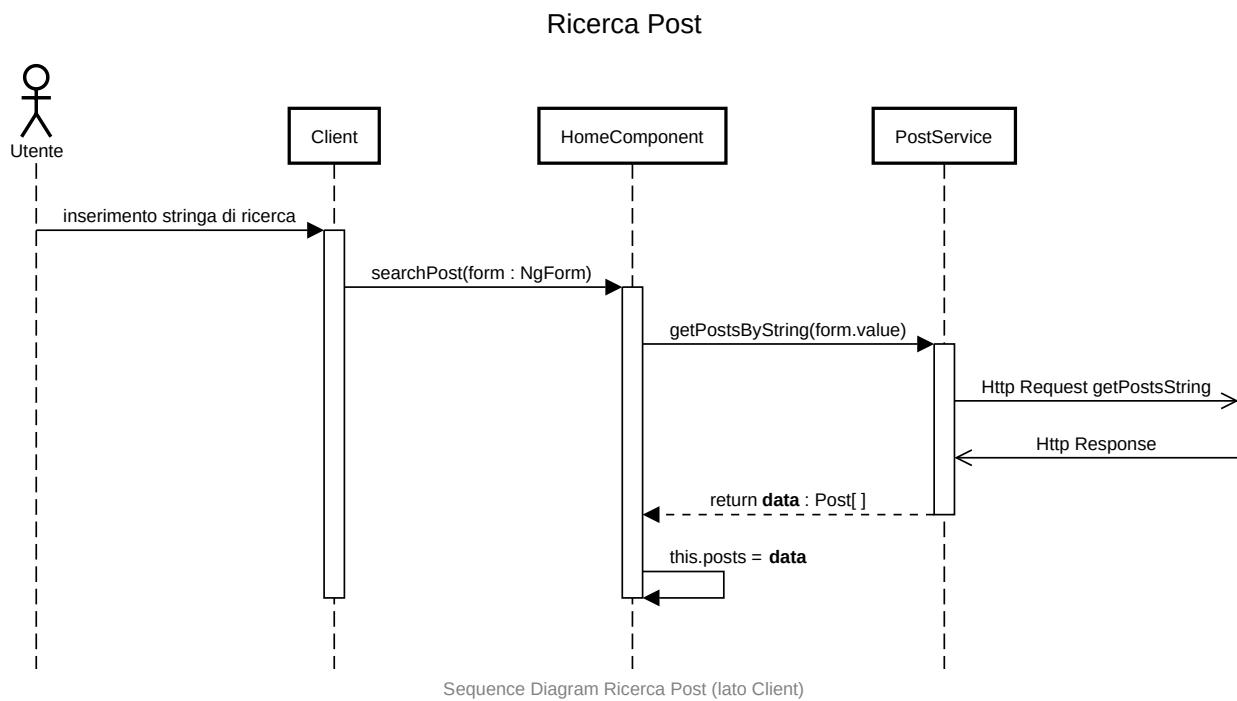
#### Sequence Diagram : EliminaCommento(Server)



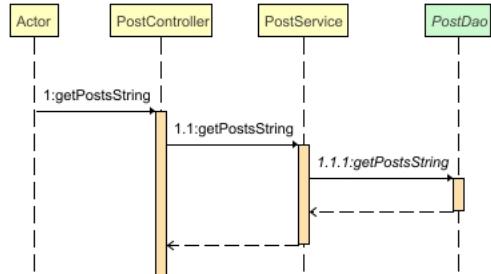
Sequence Diagram deleteCommento() (lato Server)



Sequence Diagram: RICERCA POST(Client)

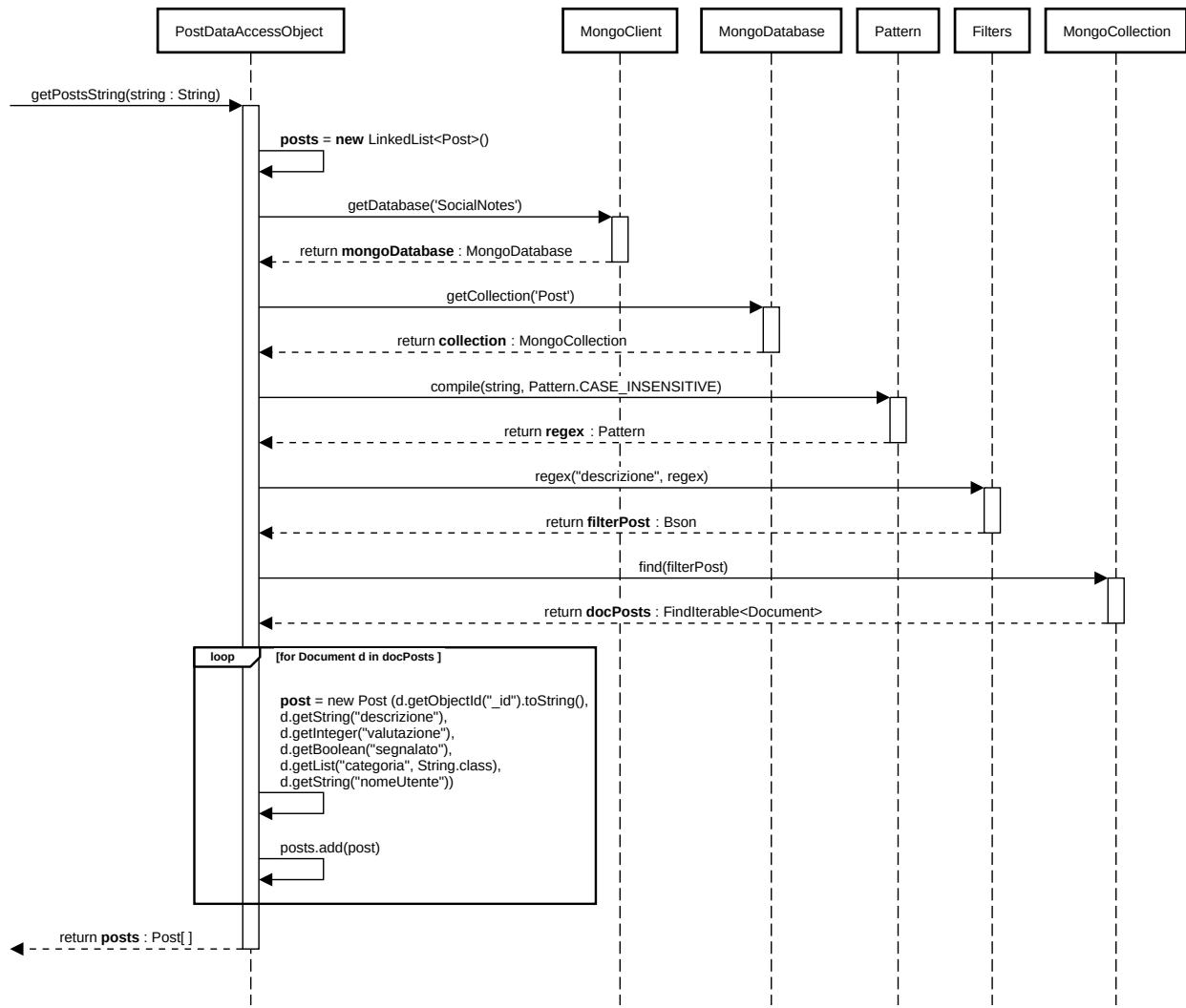


Sequence Diagram : RicercaPost(Server)

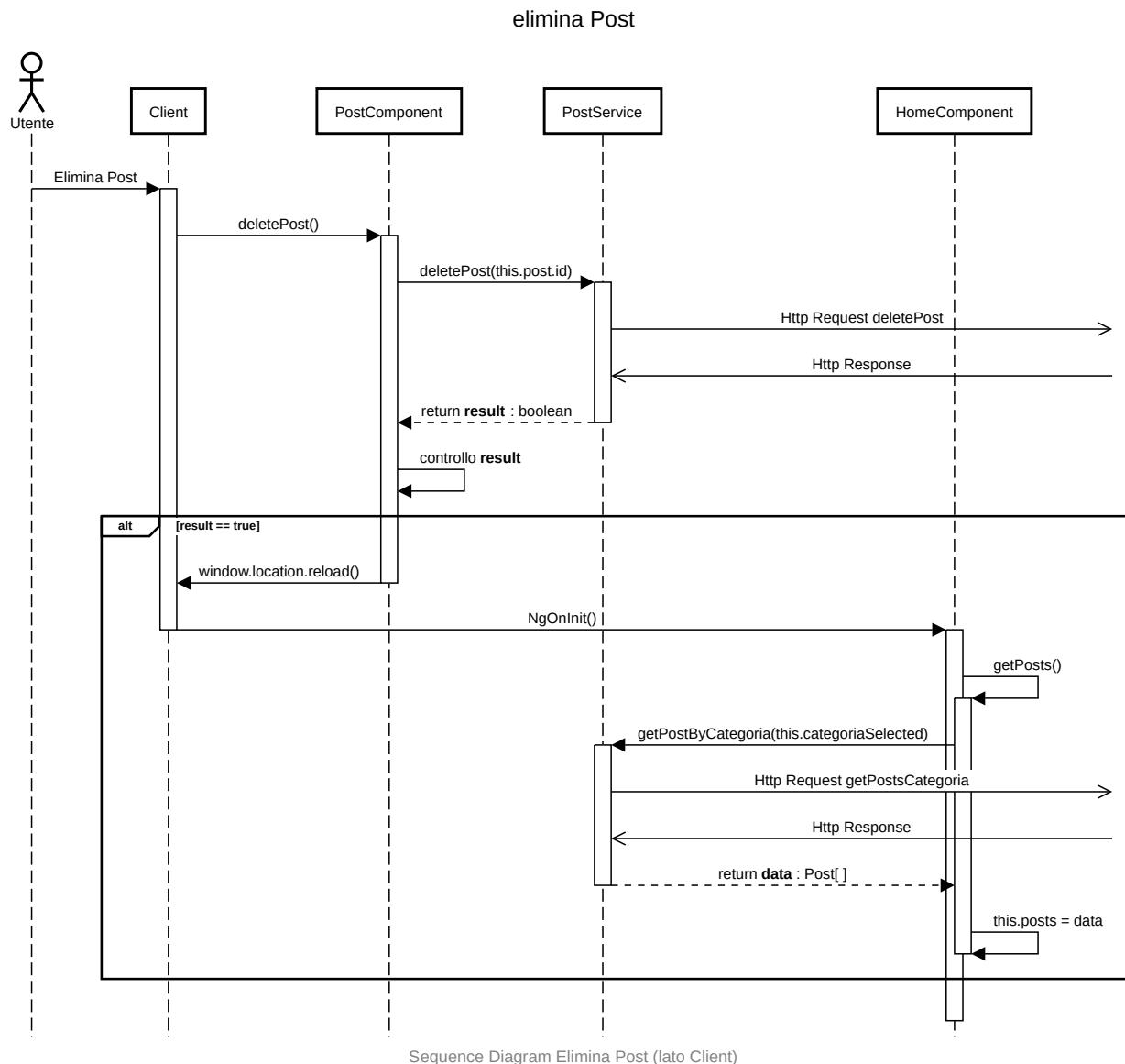


Sequence Diagram getPostsString() (lato Server)

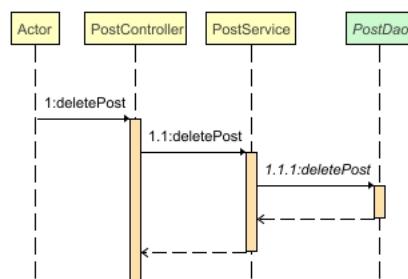
### getPostsString

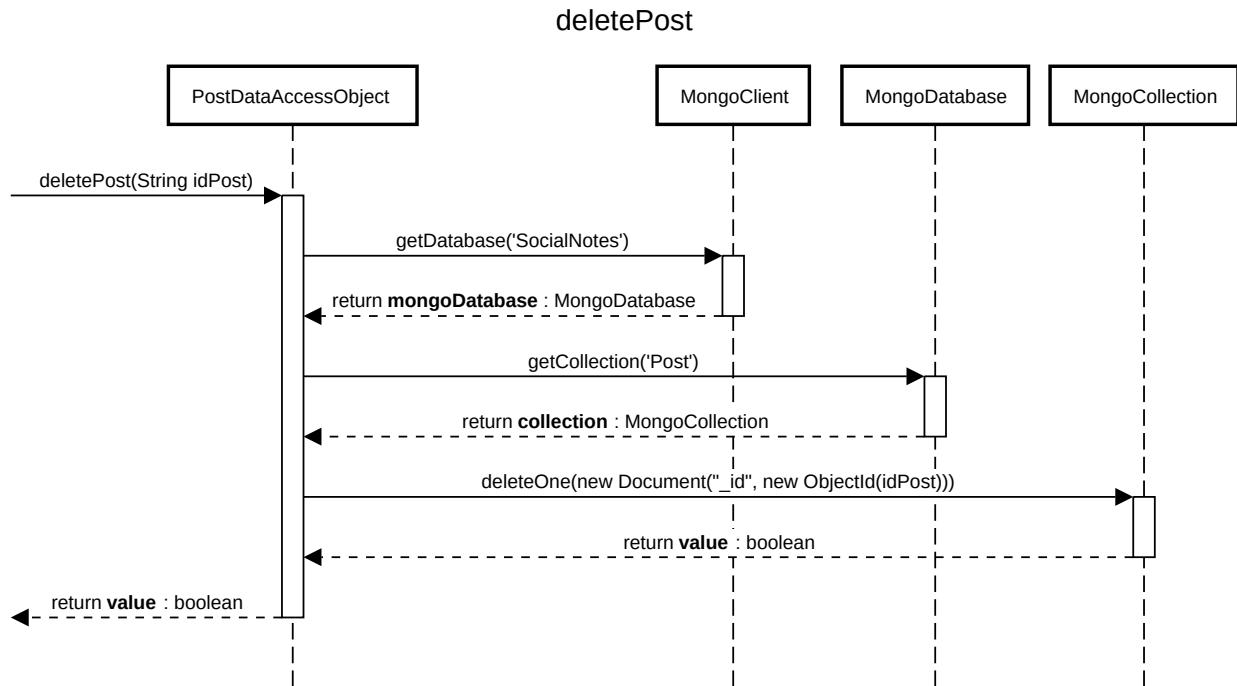


Sequence Diagram: ELIMINA POST(Client)



#### Sequence Diagram: EliminaPost(Server)





## 7 Persistenza dei dati

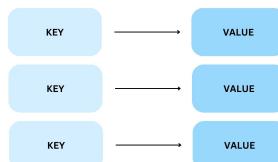
Per la realizzazione della persistenza dei dati all'interno di un database, sono stati seguiti i seguenti passi:

- Scelta del DBMS
- Individuazione delle Collection
- Creazione del database

### 7.1 Scelta del DBMS

I post pubblicati nel social Notes presentano informazioni *altamente flessibili e poco strutturate*, e che quindi poco si adattano ai database relazionali. Per questo motivo abbiamo scelto un database **NoSQL** che tipicamente non *presenta una struttura relazionale*, e dunque non usa lo schema tabulare di righe e colonne presente nella maggior parte dei sistemi di database tradizionali, usando invece un modello di archiviazione *ottimizzato per i requisiti specifici del tipo di dati* da archiviare.

Nello specifico è stato adottato **MongoDB**, un database orientato ai Documenti, in cui ogni Documento presenta una struttura con una serie di coppie chiave/valore:



I Document sono a loro volta organizzati in Collection.

## 7.2 Individuazione delle Collection

Le entità individuate nei paragrafi precedenti sono state rappresentate sotto forma di **Collection**, e come già detto, ognuna di esse contiene al suo interno i dati organizzati in Document.

Le collection individuate e implementate sono:

- **Post:** contiene le informazioni riguardanti i post di appunti pubblicati dagli utenti sulla piattaforma. Completa di:
  - id generato dal database
  - descrizione
  - media delle valutazioni assegnate dagli utenti
  - se è stato segnalato come non conforme alle regole dagli utenti
  - il nome dell'utente che ha pubblicato il post
  - le categorie a cui appartiene.
- **Utente:** contiene le informazioni riguardanti gli utenti **iscritti** alla piattaforma. Completa di:
  - id generato dal database
  - nome
  - cognome
  - nome utente, nome univoco scelto dall'utente all'atto della registrazione
  - password scelta dall'utente all'atto della registrazione
  - moderatore, che indica se l'utente è un moderatore della piattaforma o meno.
- **Commento:** contiene le informazioni riguardanti i commenti sottoscritti dagli utenti relativi ai post pubblicati sulla piattaforma. Completa di:
  - id generato dal database
  - testo inserito nel commento
  - l'id del post a cui si riferisce il commento
  - il nome utente dell'utente che ha sottoscritto il commento
- **File:** contiene le informazioni riguardanti i file pubblicati all'interno dei post, che contengono il materiale degli appunti pubblicati sulla piattaforma. Completa di:
  - id generato dal database
  - nome del file
  - estensione del file
  - id del post in cui è stato inserito il file
- **Segnalazione:** contiene le informazioni riguardanti le segnalazioni sottoscritte dagli utenti per i post. Completa di:
  - id generato dal database
  - testo inserito dall'utente
  - id del post che è stato segnalato
  - il nome utente dell'utente che ha effettuato la segnalazione
- **Valutazione:** contiene le informazioni riguardanti le valutazioni degli utenti verso i post di appunti. Completa di:
  - id generato dal database
  - valutazione inserita

- id del post valutato
- nome dell'utente che ha effettuato la valutazione

### 7.3 Costruzione del database

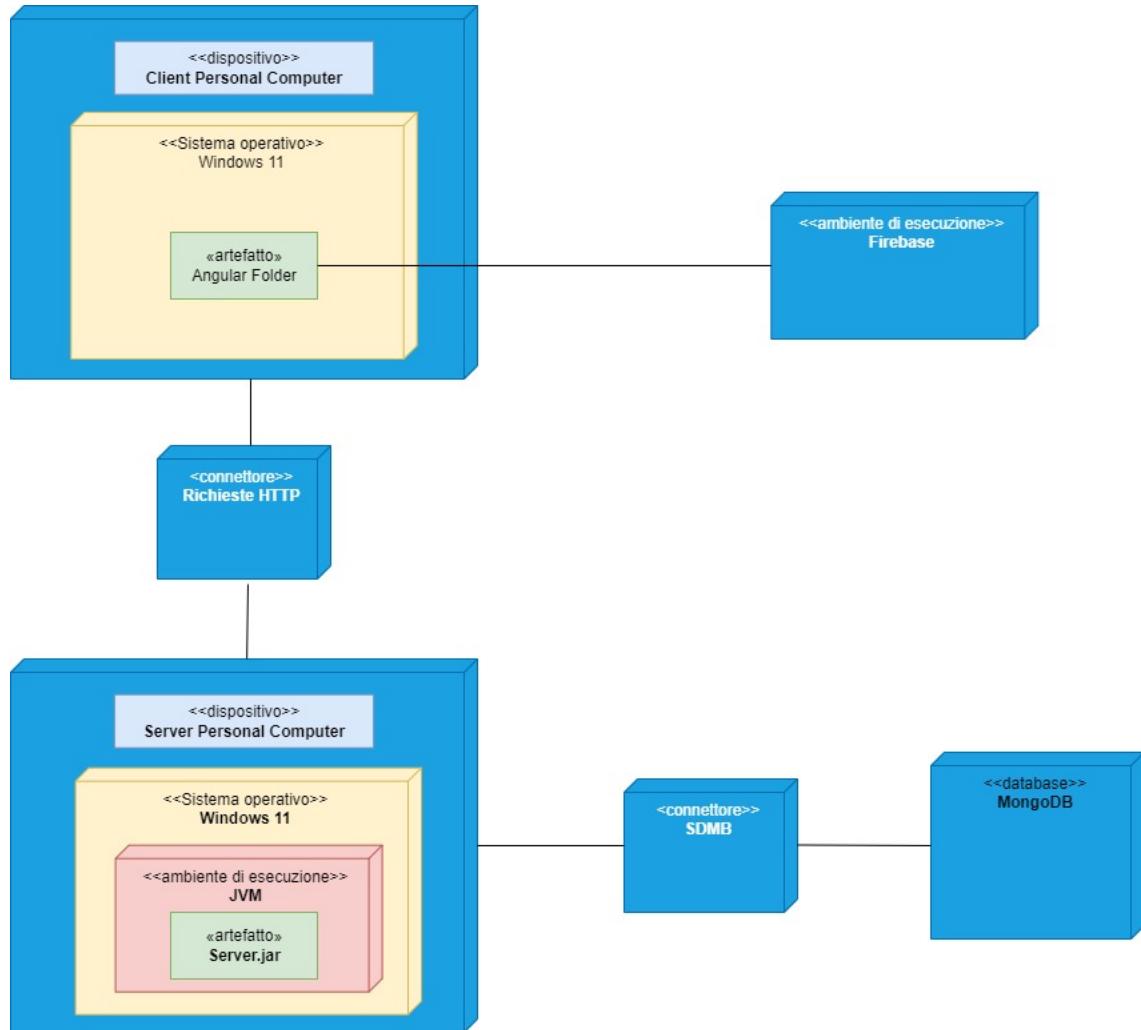
Di seguito esempi delle strutture in **Json** dei **Document** presenti all'interno delle **Collection**:



## 8 Implementazione

### 8.1 Deployment Diagram

Per rendere più chiaro il rilascio del prodotto e le modalità cui esso avviene , è sempre utile fornire un diagramma di deploy del sistema.



- Il progetto Angular, attraverso il comando `ng build`, va a creare la cartella che contiene i seguenti file:
  - Client.html : che contiene tutto il testo html del progetto
  - Client.css : che contiene tutto il codice css del progetto
  - Client.js : una traduzione in JavaScript del progetto (il quale originariamente è stato sviluppato in TypeScript)

La Angular Folder può essere resa disponibile grazie all'utilizzo di Firebase di Google. Infatti basta inizializzare un progetto Firebase all'interno della Angular Folder e poi caricarlo con una push. Così facendo Firebase renderà il progetto disponibile su internet e restituirà anche un link per accedervi.

## 9 Testing

Il **Testing** è un'attività essenziale del Ciclo di Sviluppo di un Sistema Software che deve essere eseguita *al termine di ogni iterazione* per verificare la **presenza di errori** o imperfezioni nel codice che potrebbero portare il Sistema in uno stato non previsto dalla specifica e comportarne un fallimento.

L'approccio seguito per l'attività di Testing è lo stesso che è stato impiegato nella fase di implementazione, ovvero **Bottom-Up**. Sono stati quindi testati prima i singoli componenti realizzati, i quali sono stati poi integrati gradualmente testando il sistema a seguito di ogni incremento. A sistema completo sono poi stati effettuati i test di interfaccia.

## 9.1 Testing di Unità

Con il termine Unit Testing si intende l'attività di **testing** svolta su **singole unità software**, dove per unità si intende il minimo componente di un programma dotato di funzionamento autonomo.

Lo Unit testing viene normalmente eseguito dagli sviluppatori, e può essere occasionalmente Glass Box, ovvero essere esplicitamente basato sulla conoscenza dell'architettura e del funzionamento interno di un componente (**Component Under Test - CUT**) oltre che sulle sue funzionalità esposte esternamente.

Lo Unit Test è una attività di fondamentale importanza, in quanto se eseguito in maniera scrupolosa semplifica notevolmente le successive attività di Testing, le quali potranno essere incentrate soprattutto sulla valutazione degli effetti di interazione tra i componenti montati sul sistema. Il procedimento alla base dello Unit Testing consiste nello scrivere Test Case per tutte le funzionalità e i metodi offerti dal CUT e *confrontare i risultati ottenuti con quelli attesi*.

Nome Test	Test Case ID	Descrizione	Precondizione	Input	Output attesi
Test Login	1	Test del bottone <b>LOGIN</b>	Schermata Login	1. Inserimento campi <b>Utente</b> e <b>Password</b> . 2. Click sul bottone <b>Log in</b> .	Apertura schermata <b>Home</b> .
Test Create new Account	2	Test del bottone <b>Create new Account</b>	Schermata Login	1. Click sul bottone <b>Create new Account</b> .	Apertura schermata <b>Registrazione</b> .
Test Registrazione	3	Test del bottone <b>Registrati</b>	Schermata Registrazione	1. Inserimento campi <b>Nome</b> , <b>Cognome</b> , <b>NomeUtente</b> , <b>Email</b> e <b>Password</b> . 2. Click su bottone <b>Registrati</b> .	Apertura schermata <b>Login</b>
Test LogOut	4	Test del bottone <b>Log Out</b>	Schermata Home	1. Click sul bottone <b>Log Out</b> .	Apertura Schermata <b>Login</b>
Tasto cerca post	5	Test del bottone <b>Ricerca Post</b>	Schermata Home	1. Inserimento stringa di ricerca 2. Click sul bottone <b>Ricerca</b>	Aggiornamento schermata <b>Home</b>
Test Pubblica Post	6	Test del bottone <b>Pubblica</b>	Schermata Home	1. Inserimento campi <b>Descrizione Post</b> e <b>Categoria</b> . 2. Click sul bottone <b>Pubblica</b>	Aggiornamento schermata <b>Home</b> .
Test Filtra Categoria	7	Test del bottone <b>Filtra</b>	Schermata Home	1. Inserimento campo <b>Categoria</b> .	Aggiornamento schermata <b>Home</b>
Test Elimina Post	8	Test del bottone <b>Remove post</b>	Schermata Home	1. Click sul bottone <b>Remove</b>	Aggiornamento schermata <b>Home</b>
Test Valuta Post	9	Test del bottone <b>Valuta</b>	Schermata Home	1. Inserimento campo <b>Valutazione</b> . 2. Click sul bottone <b>Valuta</b>	Aggiornamento schermata <b>Home</b>
Test Commenta Post	10	Test del bottone <b>Commenta</b>	Schermata Home	1. Inserimento campo <b>Testo</b> . 2. Click sul bottone <b>Commenta</b>	Aggiornamento schermata <b>Home</b>
Test Visualizza Commenti	11	Test del bottone <b>Visualizza Commenti</b>	Schermata Home	1. Click sul bottone <b>Visualizza Commenti</b> .	Aggiornamento schermata <b>Home</b>
Test Elimina Commento	12	Test del bottone <b>Remove Commento</b>	Schermata Home	1. Click sul bottone <b>Remove</b>	Aggiornamento schermata <b>Home</b>

## 9.2 Testing di Integrazione

Lo scopo dei Test di Integrazione è quello di integrare e combinare le unità testate (**Unit Tested**) e verificarne il comportamento come un'unità combinata attraverso le loro interfacce. Dunque i singoli moduli vengono prima testati separatamente e poi vengono integrati uno per uno, fino a quando tutti i moduli sono integrati. Quindi ne viene verificato il comportamento combinatorio e ci si assicura che i requisiti sono implementati correttamente o meno.

Le seguenti immagini mostrano il piano di test di integrazione tra server e database. Vengono effettuate delle http request al sistema, che permettono di testare il funzionamento delle funzionalità messe a disposizione dal server. Ciò è possibile grazie ad una API del Framework Spring.

#### **setPost**

Responsabile del caso d'uso **PubblicaPost**.

```
2 ▶  POST http://localhost:8080/post
3   Content-Type: application/json
4   Content-Length: 144
5   Connection: Keep-Alive
6   User-Agent: Apache-HttpClient/4.5.13 (Java/17.0.5)
7   Accept-Encoding: br,deflate,gzip,x-gzip
8
9   {
10    "descrizione": "Appunti di SAD",
11    "valutazione": 30,
12    "segnalato": false,
13    "nomeUtente": "chp"
14 }
```

Il risultato della precedente http request corrisponde al risultato atteso:

```
POST http://localhost:8080/post
Show Request

HTTP/1.1 200
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
Content-Type: application/json
Transfer-Encoding: chunked
Date: Fri, 20 Jan 2023 11:04:58 GMT
Keep-Alive: timeout=60
Connection: keep-alive

true
Response file saved.
> 2023-01-20T120458.200.json

Response code: 200; Time: 788ms (788 ms); Content length: 4 bytes (4 B)
```

#### **getPostsString**

Responsabile del caso d'uso **RicercaPost**.

```
22
23
24 ▶  GET http://localhost:8080/post/getPostsString/sad
25
26
```

Il risultato della precedente http request corrisponde al risultato atteso:

```

GET http://localhost:8080/post/getPostsString/sad
Show Request

HTTP/1.1 200
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
Content-Type: application/json
Transfer-Encoding: chunked
Date: Tue, 24 Jan 2023 14:34:45 GMT
Keep-Alive: timeout=60
Connection: keep-alive

[
  {
    "id": "63b7304dfd0880ebc3cb8c65",
    "descrizione": "Appunti di SAD",
    "valutazione": 27,
    "segnalato": true,
    "categoria": [
      "appunti"
    ],
    "nomeUtente": "giggi"
  },
  {"id": "63c6c27a5e38eb29d62a6d5b" ...},
  {"id": "63ca755a502e2703e0725369" ...}
]

```

#### **updatePostValutazione**

Responsabile del caso d'uso **ValutaPost**.

```

40
41
42 ▶ PUT http://localhost:8080/post/updatePostValutazione/63b7304dfd0880ebc3cb8c68/18/marx
43
44

```

Il risultato della precedente http request corrisponde al risultato atteso:

```

PUT http://localhost:8080/post/updatePostValutazione/63b7304dfd0880ebc3cb8c68/18/marx
Show Request

HTTP/1.1 200
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
Content-Type: application/json
Transfer-Encoding: chunked
Date: Fri, 20 Jan 2023 11:20:27 GMT
Keep-Alive: timeout=60
Connection: keep-alive

true
Response file saved.
> 2023-01-20T122027.200.json

Response code: 200; Time: 171ms (171 ms); Content length: 4 bytes (4 B)

```

### **SetUtente**

Responsabile del caso d'uso **RegistraUtente**.

```
53 ▶ POST http://localhost:8080/utente
54   Content-Type: application/json
55   Content-Length: 144
56   Connection: Keep-Alive
57   User-Agent: Apache-HttpClient/4.5.13 (Java/17.0.5)
58   Accept-Encoding: br,deflate,gzip,x-gzip
59
60   {
61     "nome": "Mario",
62     "cognome": "Rossi",
63     "nomeUtente": "marossi",
64     "email": "mario@rossi.it",
65     "password": "1234",
66     "moderatore": "false"
67 }
```

Il risultato della precedente http request corrisponde al risultato atteso:

```
POST http://localhost:8080/utente
Show Request

HTTP/1.1 200
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
Content-Length: 0
Date: Sat, 21 Jan 2023 07:30:49 GMT
Keep-Alive: timeout=60
Connection: keep-alive

<Response body is empty>

Response code: 200; Time: 874ms (874 ms); Content length: 0 bytes (0 B)
```

### **GetUtente**

Responsabile del caso d'uso **Login**.

```
75
76 ▶ GET http://localhost:8080/utente/getUtente/chp
77
```

Il risultato della precedente http request corrisponde al risultato atteso:

```

GET http://localhost:8080/utente/getUtente/chp
Show Request

HTTP/1.1 200
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sat, 21 Jan 2023 09:05:59 GMT
Keep-Alive: timeout=60
Connection: keep-alive

{
    "id": "63b73041fd0880ebc3cb8c4f",
    "nome": "Chiara",
    "cognome": "Paventa",
    "nomeUtente": "chp",
    "email": "chp@libero.it",
    "password": "fozzaGranata",
    "moderatore": true
}
Response file saved.
> 2023-01-21T100559.200.json

Response code: 200; Time: 80ms (80 ms); Content length: 156 bytes (156 B)

```

### **SetCommento**

Respondabile del caso d'uso **CommentaPost**.

```

168
169 ▶ POST http://localhost:8080/commento
170   Content-Type: application/json
171   Content-Length: 144
172   Connection: Keep-Alive
173   User-Agent: Apache-HttpClient/4.5.13 (Java/17.0.5)
174   Accept-Encoding: br,deflate,gzip,x-gzip
175
176   {
177     "testo": "Mi sono stati poco utili ai fini dell'esame",
178     "idPost": "63af7ec821df5dc651755b80",
179     "nomeUtente": "dom_98"
180   }
181

```

Il risultato della precedente http request corrisponde al risultato atteso:

```

POST http://localhost:8080/commento
Show Request

HTTP/1.1 200
Vary: Origin
Vary: Access-Control-Request-Method
Vary: Access-Control-Request-Headers
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sat, 21 Jan 2023 09:01:55 GMT
Keep-Alive: timeout=60
Connection: keep-alive

true
Response file saved.
> 2023-01-21T100155.200.json

Response code: 200; Time: 801ms (801 ms); Content length: 4 bytes (4 B)

```

### DeleteCommento

Responsabile del caso d'uso **EliminaCommento**.

```
244  
245 ► DELETE http://localhost:8080/commento/deleteCommento/63b770adfe1318d0007afb3d  
246
```

Il risultato della precedente http request corrisponde al risultato atteso:

```
DELETE http://localhost:8080/commento/deleteCommento/63b770adfe1318d0007afb3d  
Show Request  
  
HTTP/1.1 200  
Vary: Origin  
Vary: Access-Control-Request-Method  
Vary: Access-Control-Request-Headers  
Content-Type: application/json  
Transfer-Encoding: chunked  
Date: Sat, 21 Jan 2023 09:03:52 GMT  
Keep-Alive: timeout=60  
Connection: keep-alive  
  
true  
Response file saved.  
> 2023-01-21T100353.200.json  
  
Response code: 200; Time: 55ms (55 ms); Content length: 4 bytes (4 B)
```

## 9.3 Testing di interfaccia

I Test di Interfaccia hanno lo scopo di testare la User Interface. Solitamente vengono condotti in modalità **Black Box** e sono progettati per *testare il comportamento del sistema* quando sottoposto ai possibili *input forniti dall'utente sull'interfaccia*.

I casi di test sono progettati tenendo conto delle possibili interazioni che un utente può eseguire sull'interfaccia utente. A tal proposito sono state testate le seguenti interfacce :

- **La registrazione Utente** in cui l'utente inserisce i suoi dati per effettuare la registrazione: si sono testati gli input e il bottone per la registrazione e si è controllato che la registrazione fosse effettivamente eseguita dal client tramite richiesta HTTP al server, il quale la inseriva all'interno del database MongoDB, il tutto con i dati attesi dell'utente. Il sistema ha dimostrato il suo corretto funzionamento.
- **Il login Utente** : si è testato il funzionamento dei campi di login per l'inserimento delle credenziali di un utente registrato andando a testare la correttezza degli input inseriti, in questo caso il NomeUtente e la password, e il corretto funzionamento del bottone di Login tramite dei controlli che si accertassero che l'utente esistesse all'interno del sistema , che la password ad egli associato fosse corretta e che il bottone login inviasse al server i dati corretti appena inseriti dall'utente. A valle di queste operazioni il sistema ha superato tutti i casi di test, sia quelli con input corretti che non, e in tal caso mandava un messaggio di "accesso negato" all'utente .

- **La Home dell'utente:** all'interno di questo piano di test è stato testato il corretto funzionamento di tutte le funzionalità offerte all'interno di questa interfaccia grafica. In primis è stato testata l'uscita dall'account loggato tramite il bottone Log out, in cui il sistema correttamente cambia lo stato dell'utente e riporta quest'ultimo all'interfaccia di Login. In serie sono state testate l'effettivo funzionamento della pubblicazione di un post, con i suoi input, descrizione e categoria, e il tasto "Pubblica" per l'invio del post al Server; l'inserimento di un commento sotto un post con l'input per la sua descrizione e il tasto "Commenta" per la comunicazione al server ; l'inserimento della valutazione con il suo valore numerico compreso tra 1 e 30 e il tasto "Valuta" che invia al server il valore numerico atteso; anche il filtraggio dei post per categoria avviene correttamente per ogni categoria esistente così come l'eliminazione di un post da parte dell'utente loggato che ne è il possessore, il Cliente manda la richiesta di eliminazione al server che la esegue sul database in maniera univoca.

La comunicazione tra Client e Server, in tutti i casi di test sopracitati, avviene correttamente e questo è stato accertato dai messaggi di corretta esecuzione delle richieste che si è potuto constatare sulla console di ispezione del browser su cui veniva fatto girare il client e la comunicazione tra Server e Database altrettanto, poiché il server non restituiva messaggi di errori e i dati venivano inseriti correttamente all'interno del Database.

## 10 Sviluppi futuri

Nelle successive iterazioni potranno essere implementate le seguenti funzionalità:

- Introduzione di un nuovo attore, il Moderatore, per la supervisione delle policy comportamentali degli utenti all'interno della piattaforma.
- Il caso d'uso Segnalazioni post che permette di segnalare post per contenuti errati od offensivi.
- Creare un percorso educativo sotto forma di Challenge per coinvolgere e spronare gli utenti a migliorarsi sotto determinati aspetti della loro preparazione.
- Inserire degli allegati ai post di formati diversi come :pdf , foto , excel , video... .