

Università degli Studi di Napoli Federico II



Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione

Conrso di Laurea in Informatica



Loris Zannini N86003744
Mattia Marucci N86003853
Vincenzo Meloni N86003897

Anno Accademico 2023/2024

Indice

1	Introduzione	4
2	Requisiti Software	5
2.1	Analisi dei Requisiti	5
2.1.1	Requisiti del Sistema	5
2.1.2	Modellazione Casi d’Uso	6
2.1.3	Prototipazione visuale via Mock-up	8
2.1.4	Descrizioni Testuali dei Casi d’Uso	27
2.1.5	Individuazione del target degli utenti	36
2.1.6	Valutazione dell’Usabilità a priori	38
2.1.7	Glossario	42
2.2	Specifiche dei Requisiti	44
2.2.1	Classi, oggetti e relazioni di analisi	44
2.2.2	Diagrammi di sequenza di analisi	51
2.2.3	Statecharts dell’interfaccia grafica	53
3	Design del sistema	56
3.1	Descrizione dell’architettura del sistema	56
3.1.1	Sviluppo Frontend	57
3.1.2	Sviluppo Backend	58
3.1.3	API Rest	59
3.1.4	PostgreSQL	62
3.1.5	JPA Hibernate	62
3.1.6	Retrofit	62
3.1.7	Docker	62
3.1.8	Amazon EC2	63
3.2	Diagrammi di classi di design	64
3.2.1	Entità	64
3.2.2	Autenticazione	65
3.2.3	Ricerca Asta	66
3.2.4	Creazione Asta Inversa	67
3.2.5	Creazione Asta al Ribasso	68
3.2.6	Creazione Asta Silenziosa	69
3.2.7	Gestione Notifiche	70
3.2.8	Personalizzazione Profilo	71
3.2.9	Presentazione Offerta	72
3.3	Diagrammi di sequenza di design	73
3.3.1	Creazione Asta Silenziosa	74
3.3.2	Presentazione Offerta Asta Silenziosa	75

3.3.3	Ricerca Asta	76
3.3.4	Visualizzazione Profilo Creatore Asta	77
4	Codice sorgente	78
4.1	File di Build automatica	78
4.2	Versioning	81
4.3	Report di qualità del codice	81
5	Testing e Valutazione dell’usabilità	82
5.1	Test JUnit	82
5.1.1	calcolaTempoRimanente	83
5.1.2	passwordValida	85
5.1.3	trovaAstePerParolaChiaveAndCategoria	87
5.1.4	verificaOfferta	89
5.2	Valutazione dell’usabilità sul campo	91
5.2.1	Test di usabilità	91
5.2.2	Report Google Analytics	92

1 Introduzione

DietiDeals24 è una piattaforma che si occupa della gestione di aste online; consiste in un'applicazione mobile performante e affidabile, progettata per offrire un'esperienza intuitiva, rapida e piacevole agli utenti. L'obiettivo principale di tale software è permettere la compravendita di qualsiasi oggetto tramite un sistema di aste dotato di un'interfaccia concisa e facilmente navigabile, consentendo una gestione efficiente e ottimizzata anche per utenti inesperti. Il sistema offre la possibilità di entrare con due tipi di account differenti: venditore e compratore, a seconda delle nostre esigenze. Al primo accesso, l'utente potrà registrare il proprio account e scegliere il tipo di account con cui entrare (può essere usata la stessa e-mail, per al più un account venditore ed uno compratore). DietiDeals24, oltre a garantire la vendita di beni tramite un sistema di aste che si differenziano in 3 tipologie, permette una ricerca approfondita tra le aste attive con la possibilità di filtrare per tipo di asta, categoria e/o parola chiave; in più un usufruitore può gestire e personalizzare la propria pagina profilo, ed esplorare quelle altrui, potendo visualizzare tutte le aste create, e quelle vinte, di ogni singolo utente. Un venditore può creare due tipi di asta: al ribasso o silenziosa. Un'asta al ribasso è caratterizzata da un prezzo iniziale stabilito dal venditore, da un timer per decrementare il prezzo, da un importo per ciascun decremento e da un prezzo minimo segreto a cui vendere il prodotto. Il raggiungimento del prezzo minimo (tramite il decremento), porterà al fallimento dell'asta; il primo compratore a presentare un'offerta vincerà l'asta. In un'asta silenziosa, invece, il venditore specifica una data di scadenza e i compratori possono inviare offerte segrete al venditore, che potranno essere accettate o rifiutate. L'offerta accettata sarà quella vincente. Un compratore può creare un solo tipo di asta: quella inversa. In questa tipologia, il compratore specifica il prodotto richiesto, un prezzo iniziale che è disposto a pagare e una data di scadenza. I venditori che dispongono di quel determinato prodotto possono partecipare all'asta presentando offerte al ribasso. Al momento della scadenza dell'asta, il venditore con l'offerta più bassa vince. DietiDeals24, dunque, rappresenta un software efficiente in grado di consentire lo scambio di prodotti tramite un sistema di compravendita innovativo che dà la possibilità, al cliente, di poter generare e partecipare ad aste di diverso tipo in base alle proprie necessità.

2 Requisiti Software

2.1 Analisi dei Requisiti

Nell'analisi dei requisiti vengono mostrate le funzionalità e le interfacce che il sistema deve soddisfare, per garantire che il prodotto finale risponda alle esigenze degli utenti.

2.1.1 Requisiti del Sistema

Di seguito sono elencati i requisiti funzionali del sistema, ottenuti da un'analisi dei requisiti utente:

1. Autenticazione

Il sistema permette ad ogni utente di registrarsi e/o effettuare il login; L'autenticazione può essere effettuata anche mediante Google.

2. Ricerca aste

Il sistema permette ad ogni utente di cercare tra le aste presenti nel sistema, con la possibilità di filtrare la ricerca per categoria, parola chiave o entrambe.

3. Creazione aste

Il sistema permette, in base al tipo dell'utente, di poter creare tre tipi differenti di aste: inversa, al ribasso o silenziosa.

4. Presentazione offerte

Il sistema permette ad ogni utente di presentare offerte per partecipare ad aste. Le offerte sono gestite in modo differente in base al tipo di asta.

5. Personalizzazione profilo

Il sistema permette ad ogni utente di personalizzare il proprio profilo, aggiungendo foto, short bio, sito web e paese, o modificare la propria password. Un utente può visualizzare il profilo del creatore di un'asta.

6. Gestione notifiche

Il sistema dispone di una sezione dedicata alle notifiche, in cui ogni utente può visualizzarle e rimuoverle. Un utente riceve una notifica quando si aggiudica un'asta o quando la sua asta creata si conclude.

2.1.2 Modellazione Casi d'Uso

Diagramma dei casi d'uso Autenticazione -

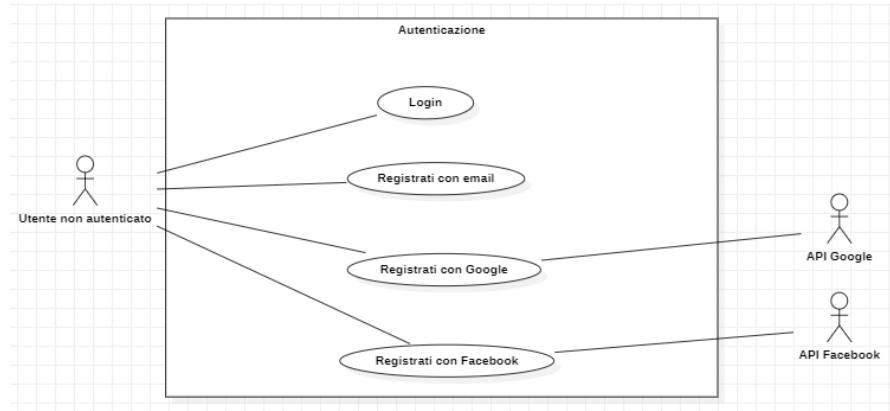
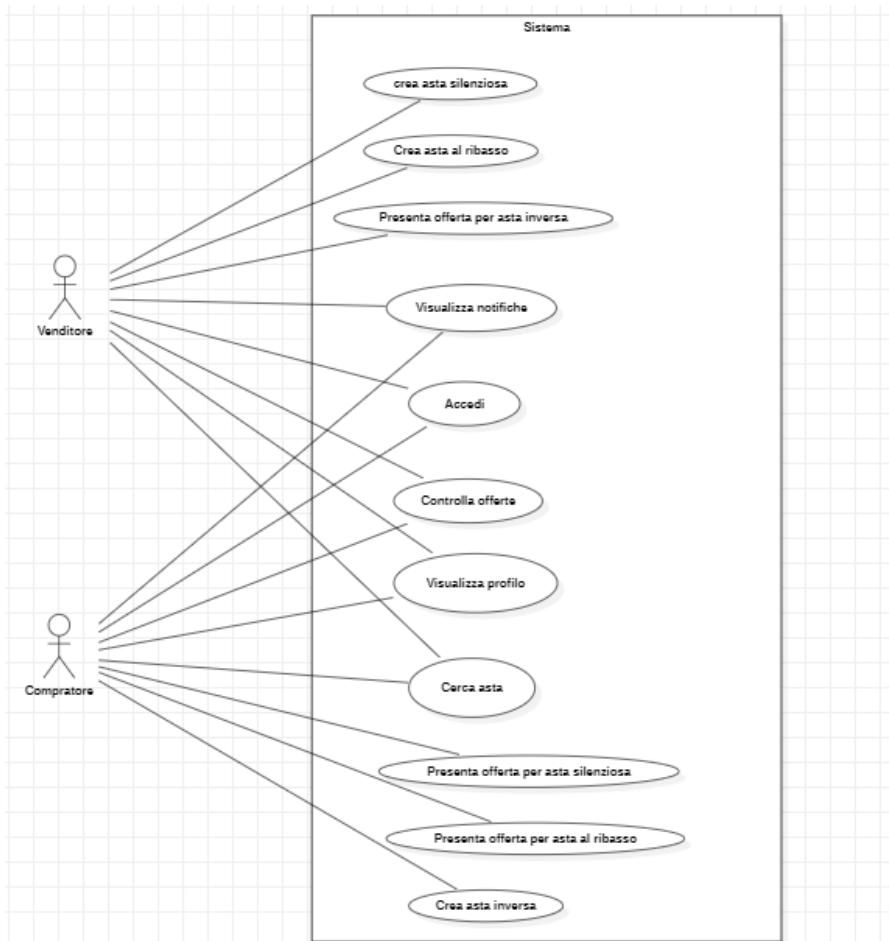


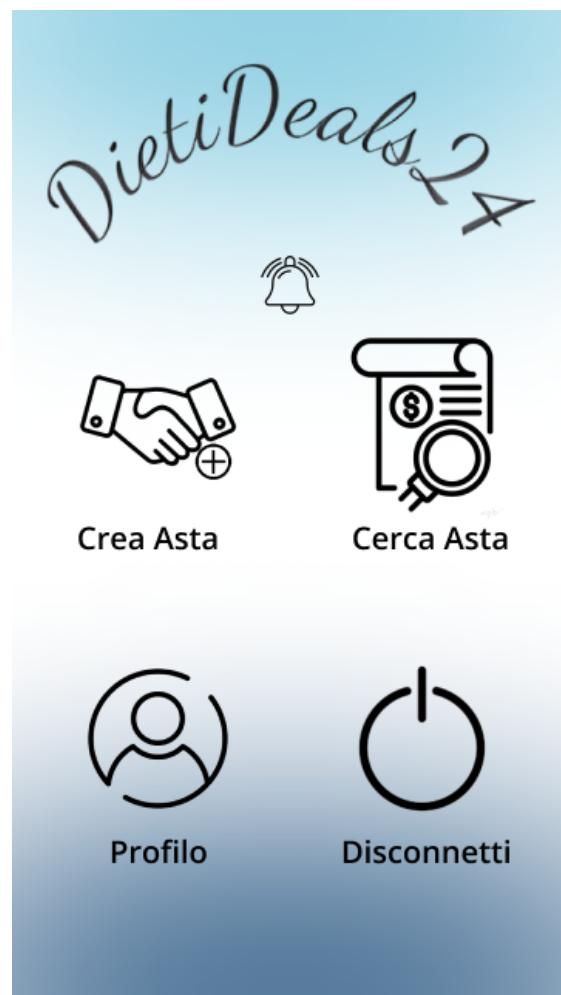
Diagramma dei casi d'uso Interazione con il sistema -



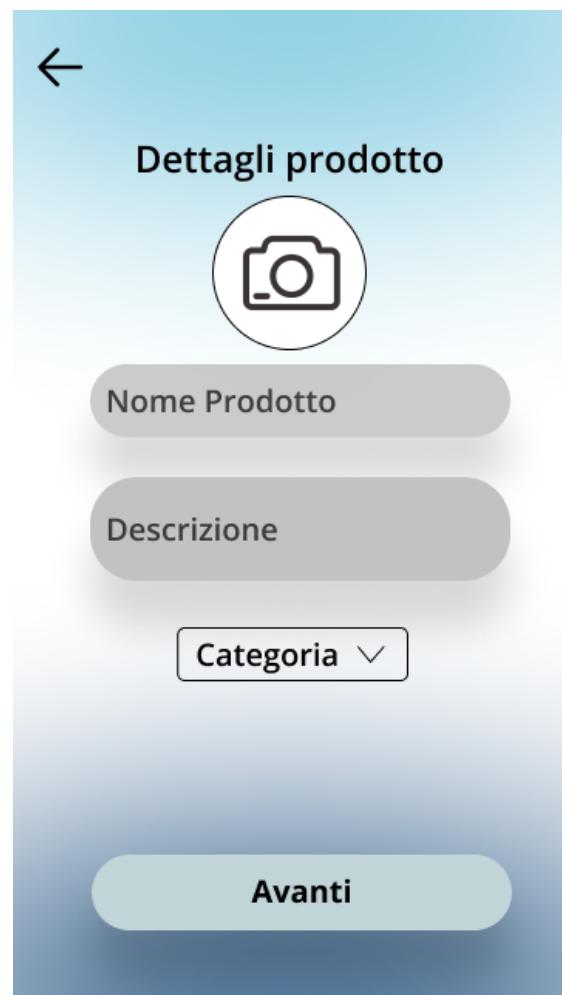
2.1.3 Prototipazione visuale via Mock-up

In questa sezione verranno mostrati i Mock-up necessari alla comprensione dei casi d'uso. Lo strumento utilizzato per la creazione delle interfacce è Figma. Il progetto completo è disponibile al seguente link.

Caso d'Uso #1: Creazione asta silenziosa



Mock-up Home



Mock-up Crea asta



Mock-up Scelta tipo asta



Mock-up Crea asta silenziosa

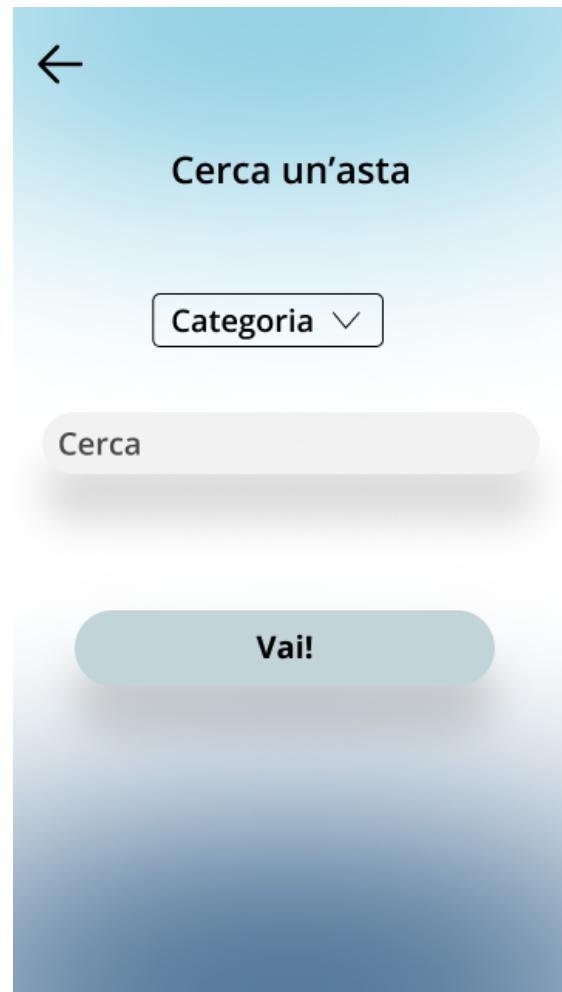


Mock-up Asta silenziosa creata

Caso d'Uso #2: Presentazione offerta per asta silenziosa



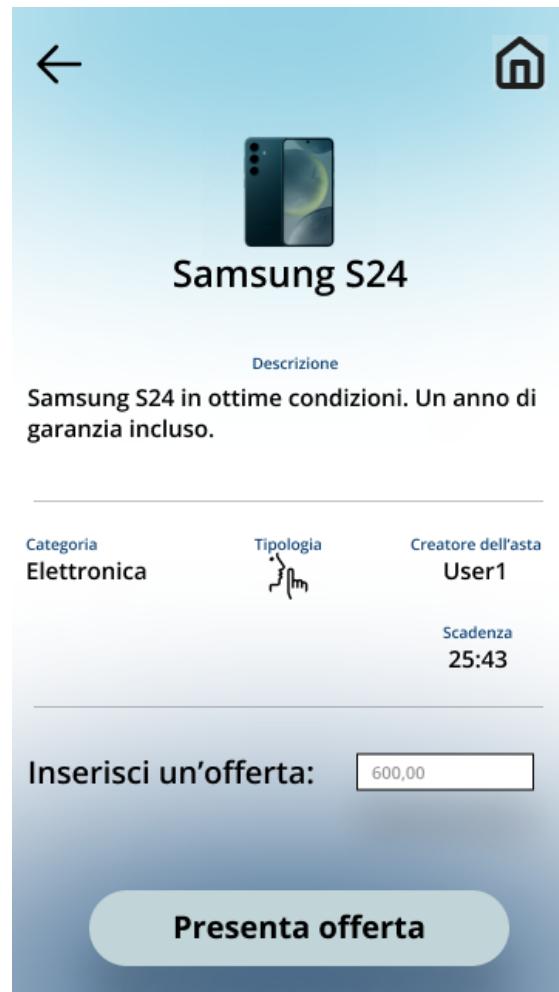
Mock-up Home



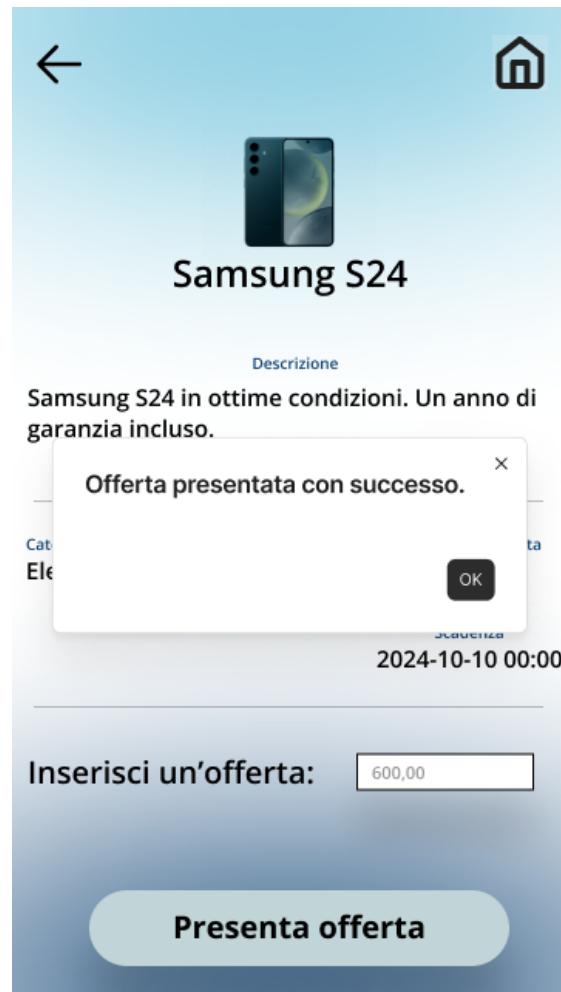
Mock-up Cerca asta



Mock-up Risultati ricerca



Mock-up Scheda prodotto asta silenziosa

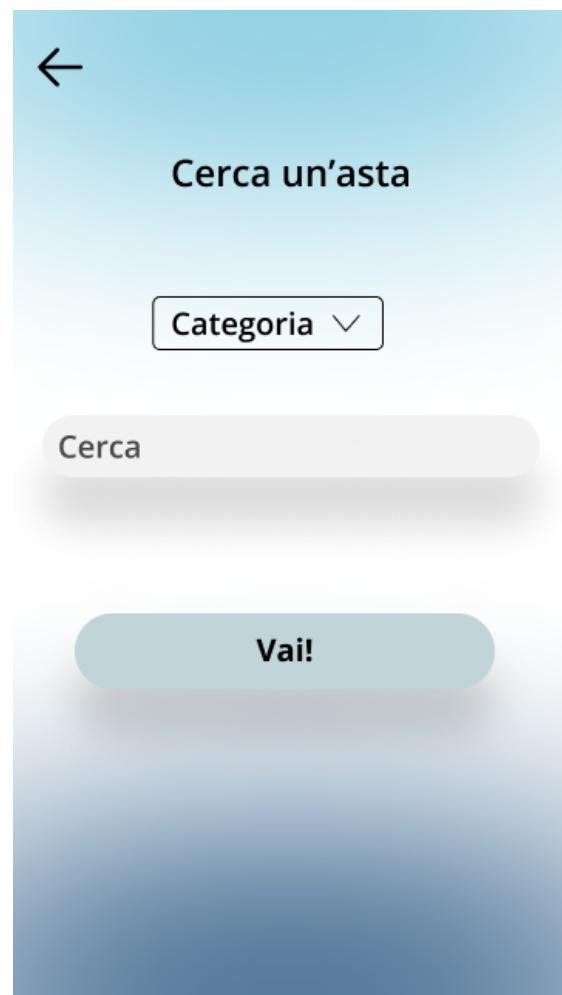


Mock-up Offerta presentata asta silenziosa

Caso d'Uso #3: Ricerca asta



Mock-up Home



Mock-up Cerca asta



Mock-up Risultati ricerca

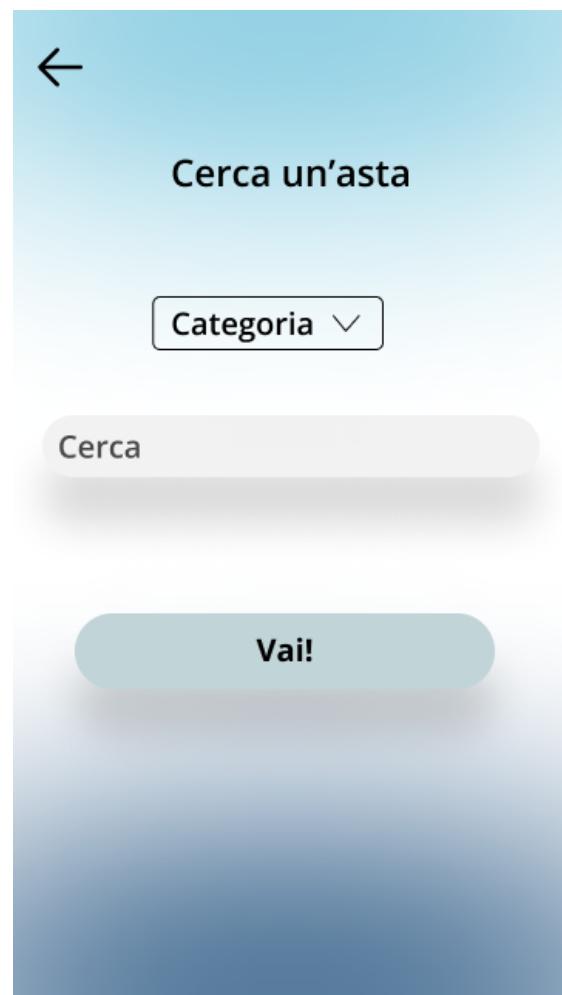


Mock-up Scheda prodotto asta al ribasso

Caso d'Uso #4: Visualizzazione profilo creatore asta



Mock-up Home



Mock-up Cerca asta



Mock-up Risultati ricerca



Mock-up Scheda prodotto asta al ribasso



Mock-up Profilo creatore asta

2.1.4 Descrizioni Testuali dei Casi d'Uso

Dare una descrizione dettagliata su come deve agire l'utente e rispondere il sistema in un determinato scenario serve per descrivere come l'applicativo deve comportarsi in scenari reali, garantendo una comprensione comune delle funzionalità del sistema.

In questa sezione verranno descritti i seguenti casi d'uso:

1. Creazione asta silenziosa
2. Presentazione offerta per asta silenziosa
3. Ricerca asta
4. Visualizzazione profilo creatore asta

Use Case #1	Crea asta silenziosa		
Goal	Il venditore vuole creare una nuova asta per un determinato prodotto/servizio per provare a venderlo al miglior offerente		
Preconditions	Il venditore deve aver effettuato il login come venditore con successo		
Success End Conditions	L'asta creata dal venditore viene resa disponibile per tutti gli utenti		
Description	Step	Attore Venditore	Sistema
	1	Preme sul bottone “Crea Asta” nella pagina Home	
	2		Mostra pagina Crea Asta
	3	Inserisce i campi dell'asta nome, descrizione, foto, categoria	
	4	Preme sul bottone “Crea”	
	5		Mostra pagina Scelta tipo
	6	Preme sul bottone asta silenziosa	
	7		Mostra pagina Crea Asta Silenziosa

	8	Inserisce una scadenza valida	
	9	Preme sul bottone "Crea"	
	10		Crea l'asta assegnandole un'id univoco e l'id dell'utente che l'ha creata
Extensions	Step	Attore Venditore	Sistema
A Il venditore torna indietro da Crea Asta	3.a	Preme sul bottone indietro nella pagina Crea Asta	
		Ricomincia dal passo 1 del main scenario	
B Il venditore lascia nome e/o categoria vuoti	4.b		Mostra Errore campi non compilati in Crea Asta
		Ricomincia dal passo 3 del main scenario	
C Il venditore torna indietro da Crea Asta Silenziosa	8.c a 9.c	Preme sul bottone indietro	
	10.c		Mostra pagina Scelta tipo
		Ricomincia dal passo 6 del main scenario	
SUB-VARIATIONS	Step	Attore Venditore	Sistema
D Il venditore crea la asta dal menu di visualizzazione delle proprie aste	1.d	Preme sul bottone profilo nella pagina Home	
	2.d		Mostra pagina Profilo
	3.d	Preme sul bottone visualizza asta create nella pagina Profilo	
	4.d		Mostra messaggio Nessuna asta creata
	5.d	Preme sul bottone "Crea" nella pagina Aste create	

	Ricomincia dal passo 2 del main scenario
--	--

Table 1: Descrizione Testuale Use Case #1

Use Case #2	Presenta offerta per un'asta silenziosa (compratore)		
Goal	Il compratore vuole presentare un'offerta segreta per una determinata asta		
Preconditions	Il compratore deve aver effettuato il login come compratore con successo		
Success End Conditions	L'offerta fatta dal compratore arriva correttamente al venditore che ha creato quell'asta		
Description	Step	Attore Compratore	Sistema
	1	Preme sul bottone “Cerca Asta” nella pagina Home	
	2		Mostra pagina Cerca Asta
	3	Seleziona “Filtro” e inserisce il prodotto/categoria da cercare	
	4	Preme sul bottone “Cerca”	
	5		Mostra pagina Risultati Ricerca
	6	Preme sul bottone relativo all'asta corrispondente	
	7		Mostra pagina Scheda prodotto
	8	Inserisce un importo per indicare l'offerta che vuole fare	

	9	Preme sul bottone "Presenta offerta" nella pagina Scheda prodotto	
	10		Invia l'offerta al venditore che ha creato quell'asta
Extensions	Step	Attore Compratore	Sistema
A Il compratore annulla la ricerca	3.a a 4.a	Preme sul bottone indietro nella pagina Cerca Asta	
		Ricomincia dal passo 1 del main scenario	
B Il compratore non inserisce un nome	5.b		Mostra Errore campi non compilati in Cerca Asta
		Ricomincia dal passo 3 del main scenario	
C La ricerca non produce risultati	5.c		Mostra Nessun risultato ricerca in Risultati Ricerca
	6.c	Preme sul bottone indietro nella pagina Scheda prodotto	
		Ricomincia dal passo 2 del main scenario	
D L'asta per il prodotto selezionato non è più attiva	7.d		Mostra Errore asta scaduta
	8.d	Preme sul bottone indietro nella pagina Scheda prodotto	
		Ricomincia dal passo 5 del main scenario	
E Il compratore torna indietro da Risultati Ricerca	6.e	Preme sul bottone indietro nella pagina Risultati Ricerca	
		Ricomincia dal passo 2 del main scenario	
F Il compratore torna indietro mentre visualizza il prodotto	8.f	Preme sul bottone indietro nella pagina Scheda prodotto	

	Ricomincia dal passo 5 del main scenario		
G Il compratore inserisce un importo non valido	10.g		Mostra Errore importo non valido offerta
	Ricomincia dal passo 7 del main scenario		
SUB-VARIATIONS	Step	Attore Compratore	Sistema
H Il compratore presenta una nuova offerta dalla pagina Aste Seguite	1.h	Preme sul bottone profilo nella pagina Home	
	2.h		Mostra pagina Profilo
	3.h	Preme sul bottone visualizza aste seguite nella pagina Profilo	
	4.h		Mostra pagina Aste Seguite
	5.h	Preme su un'asta tra quelle attive	
	Ricomincia dal passo 7 del main scenario		

Table 2: Descrizione Testuale Use Case #2

Use Case #3	Ricerca Asta		
Goal	Un utente registrato (compratore o venditore) vuole cercare un'asta		
Preconditions	L'utente deve aver effettuato il login come compratore o venditore con successo		
Success End Conditions	L'utente registrato riesce a vedere la lista di una o più aste che corrispondono alla propria ricerca		
Description	Step	Attore Utente registrato	Sistema
	1	Preme sul bottone “Cerca Asta” nella pagina Home	
	2		Mostra pagina Cerca Asta
	3	Seleziona “Filtro” e inserisce il prodotto/categoria da cercare	
	4	Preme sul bottone “Cerca”	
	5		Mostra pagina Risultati Ricerca
	6	Preme sul bottone relativo all'asta corrispondente	
Extensions	7		Mostra pagina Scheda prodotto
	Step	Attore Utente registrato	Sistema
A L'utente annulla la ricerca	3.a a 4.a	Preme sul bottone indietro nella pagina Cerca Asta	
		Ricomincia dal passo 1 del main scenario	
B L'utente non inserisce un nome	5.b		Mostra Errore campi non compilati in Cerca Asta
		Ricomincia dal passo 3 del main scenario	
C La ricerca non produce risultati	4.c		Mostra Nessun risultato ricerca in Risultati Ricerca

	5.c	Preme sul bottone indietro	
	Ricomincia dal passo 2 del main scenario		
D L'utente torna indietro mentre visualizza l'asta	6.d	Preme sul bottone con la freccia per tornare indietro nella pagina Scheda prodotto	
	Mostra pagina Risultati Ricerca		
SUB-VARIATIONS	Step	Attore Utente registrato	Sistema
E L'utente cerca un'asta da Aste Seguite	1.e	Preme sul bottone profilo nella pagina Home	
	2.e		Mostra pagina Profilo
	3.e	Preme sul bottone visualizza aste seguite nella pagina Profilo	
	4.e		Mostra pagina aste seguite
	5.e		Mostra messaggio nessun asta presente
	6.e	Preme sul bottone "Cerca"	
	Ricomincia dal passo 5 del main scenario		

Table 3: Descrizione Testuale Use Case #3

Use Case #4	Visualizza profilo di un creatore di un'asta
Goal	Un utente registrato (compratore o venditore) vuole visualizzare le informazioni sul profilo di un altro utente
Preconditions	L'utente deve aver effettuato il login come compratore o venditore con successo
Success End Conditions	L'utente registrato riesce a vedere le informazioni di un altro utente (email, short bio, sito web, ecc.)

	Step	Attore Utente registrato	Sistema
Description	1	Preme sul bottone “Cerca Asta” nella pagina Home	
	2		Mostra pagina Cerca Asta
	3	Seleziona “Filtro” e inserisce il prodotto/categoria da cercare	
	4	Preme sul bottone “Cerca”	
	5		Mostra pagina Risultati Ricerca
	6	Preme sul bottone relativo all'asta corrispondente	
	7		Mostra pagina Scheda prodotto
	8	Clicca sul nome del creatore dell'asta	
	9		Mostra pagina Profilo
Extensions	Step	Attore Utente registrato	Sistema
A L'utente annulla la ricerca	3.a a 4.a	Preme sul bottone indietro nella pagina Cerca Asta	
		Ricomincia dal passo 1 del main scenario	
B L'utente non inserisce un nome	5.b		Mostra Errore campi non compilati in Cerca Asta
		Ricomincia dal passo 3 del main scenario	
C La ricerca non produce risultati	4.c		Mostra Nessun risultato ricerca in Risultati Ricerca
	5.c	Preme sul bottone indietro	

		Ricomincia dal passo 2 del main scenario	
D L'utente va indietro dopo la ricerca	6.d	Preme sul bottone con la freccia per tornare indietro	
	Ricomincia dal passo 5 del main scenario		
Sub-variations	Step	Attore Utente registrato	Sistema
E L'utente visualizza la Scheda prodotto dalle Aste Seguite	1.e	Preme sul bottone profilo nella pagina Home	
	2.e		Mostra pagina Pro- filo
	3.e	Preme sul bottone visualizza aste se- guite nella pagina Profilo	
	4.e		Mostra pagina Aste Seguite
	5.e	Preme sul bottone relativo all'asta da selezionare	
	Ricomincia dal passo 7 del main scenario		
F L'utente visualizza la Scheda prodotto dalle Notifiche	1.f	Preme sul bottone notifiche nella pag- ina Home	
	2.f		Mostra pagina No- tifiche
	3.f	Visualizza una no- tifica riguardante un'asta a cui sta partecipando e preme sul nome dell'asta	
	Ricomincia dal passo 7 del main scenario		

Table 4: Descrizione Testuale Use Case #4

2.1.5 Individuazione del target degli utenti

Individuare il target di utenti è un passo fondamentale nell'ambito di sviluppo di un software, poichè consiste nel comprendere chi sono gli utenti, quali sono le loro esigenze, preferenze e abitudini di utilizzo, e in quale contesto utilizzeranno il software. Questo processo permette di creare una documentazione e un'interfaccia grafica che rispondano efficacemente alle necessità specifiche di ciascun gruppo di utenti, migliorando così l'usabilità e la soddisfazione complessiva degli usufruitore della piattaforma. Per quanto riguarda il software DietiDeals24, analizzando i vari contesti, abbiamo identificato due gruppi di utenti:

- 1. Venditore**
- 2. Compratore**

Sia venditori che compratori potrebbero essere privati o professionali, dunque da ciò si deduce che un software di questo tipo può raccogliere un bacino di utenza elevato, formato ad esempio da venditori e compratori occasionali, ma anche da utenti che lavorano quotidianamente nel settore della compra-vendita. Inoltre, ci sarà una fetta di utenti che vorrà sia vendere che comprare articoli: ciò sarà permesso nel nostro sistema poichè ogni utente potrà entrare sia come venditore che come compratore.

Dall'individuazione del target di utenti, abbiamo definito 3 modelli "Personas", i quali rappresentano rappresentazioni fittizie di utenti ideali create sulla base delle precedenti analisi:



Anna

- Età: 42
- Luogo: Milano
- Stato: Italia
- Sesso: Donna

Bio

Anna è una proprietaria di un negozio di antiquariato a Milano, con una passione per gli oggetti d'epoca e un'ampia conoscenza del settore. Ha scoperto le aste online come un modo efficace per vendere i suoi articoli d'antiquariato a una clientela più ampia. Anna ha buone conoscenze tecnologiche e si avvale del suo tablet per la gestione e il monitoraggio delle aste quando è fuori dal negozio. Ama personalizzare il proprio profilo per mostrare agli utenti i suoi interessi.

Personalità



Persona 1: Anna



Gianluca

- Età: 23
- Luogo: Roma
- Stato: Italia
- Sesso: Uomo

Bio

Gianluca è uno studente universitario che vive a Roma. È molto appassionato di libri e fumetti, e spesso cerca modi per risparmiare sui suoi acquisti attraverso le aste online. Utilizza principalmente il suo smartphone per partecipare alle aste, poiché è sempre in movimento tra lezioni e studio. Gianluca ha una bassa competenza tecnologica, per questo motivo vuole trovare una applicazione funzionale con una interfaccia minimale e concisa, che gli permetta di riuscire a navigare tra le aste e presentare le offerte senza problemi.

Personalità



Persona 2: Gianluca



Marco

- Età: 34
- Luogo: Pisa
- Stato: Italia
- Sesso: Uomo

Bio

Marco è un impiegato amministrativo che vive a Pisa. Ama esplorare nuove tecnologie e gadget, e nel suo tempo libero si diletta nella ricerca di buone aste online. Non è un esperto di aste, ma ha scoperto che può trovare ottimi affari su oggetti di interesse, come elettrodomestici e gadget tecnologici. Ha una competenza tecnologica media e utilizza principalmente il suo tablet a casa e lo smartphone quando è in giro. Le notifiche sono essenziali per lui, poiché non essendo molto attivo può comunque tenere traccia delle sue offerte.

Personalità



Persona 3: Marco

2.1.6 Valutazione dell'Usabilità a priori

La valutazione dell'usabilità a priori in una documentazione software serve a garantire che la documentazione sia efficace, comprensibile e utilizzabile prima che venga rilasciata agli utenti finali.

In questa fase, vengono applicati principi euristici o determinate regole che permettano di analizzare l'interfaccia del sistema al fine di identificare le potenziali dinamiche che possano creare problemi all'interazione dell'utente con l'applicazione. Il modello su cui ci siamo basati riguarda le **otto regole d'oro di Ben Shneiderman**, uno dei più importanti informatici della sua epoca, nonché pionere dell'interazione uomo-computer. Alcune delle euristiche ampiamente conosciute nel campo dell'usabilità sono state formulate da Ben Shneiderman, uno dei pionieri dell'interazione uomo-macchina. Le Gold Rules sono linee guida fondamentali per la progettazione di interfacce utente efficaci ed usabili. Di seguito elenchiamo le otto regole:

1. Coerenza a tutti i costi

All'interno del sistema verranno usate terminologie, layout, colori, e comportamenti uniformi in tutta l'interfaccia. Dare coerenza aiuta gli utenti a prevedere cosa accadrà in seguito.

2. Usabilità universale

L’interfaccia utente avrà scorciatoie, comandi rapidi o macro per le operazioni comuni, riducendo il tempo e gli sforzi per gli utenti esperti.

3. Offrire riscontri informativi

Ad ogni azione ci sarà una risposta dell’interfaccia, in modo tale che ogni utente abbia piena conoscenza su ciò che sta succendendo in ogni momento.

4. Dialogo con gli utilizzatori

Le sequenze di azioni devono essere organizzate con un’inizio, un punto intermedio e una conclusione. Questo dà agli utenti un senso di sollievo e di raggiungimento degli obiettivi.

5. Prevenire gli errori

Prevenire gli errori è fondamentale per offrire una buona esperienza all’utente. Per questo motivo nell’applicazione verranno utilizzate interfacce create in modo tale da poter dare all’utente un range di possibilità più ristretto, che riguardi solo ciò che riguarda il dominio del problema.

6. Assicurare la versatilità

Per quanto possibile, le azioni dovrebbero essere reversibili. In questo modo gli utenti sono consapevoli che gli errori possono essere annullati.

7. Garantire il controllo agli utenti

Gli utenti dovrebbero sentirsi al controllo delle loro azioni e del sistema. Il design andrà a minimizzare le azioni imprevedibili e darà agli utenti la sensazione di padroneggiare l’interfaccia.

8. Ridurre il calcolo di memoria a breve termine

Verranno utilizzate interfacce che minimizzino la necessità di ricordare informazioni tra diverse parti del sistema, evitando che l’utente debba memorizzare informazioni da riportare da una schermata ad un’altra.

Test di usabilità

Abbiamo completato la parte di valutazione dell'usabilità a priori eseguendo un Test di usabilità, il cui scopo è quello di assegnare delle task da completare tramite i mock up Figma e osservare i risultati.

Nello specifico abbiamo preso in considerazione i seguenti valutatori:

1. **Maria**

Bassa conoscenza tecnologica e del dominio dell'applicazione.

2. **Luca**

Alta conoscenza tecnologica e del dominio dell'applicazione.

3. **Francesco**

Bassa conoscenza tecnologica e alta conoscenza del dominio dell'applicazione.

Sono state effettuate le seguenti **task**:

1. Registrazione nel sistema
2. Creazione di un'asta silenziosa
3. Presentazione di un'offerta

L'**esito** di ogni task sarà:

1. **S (Success)**, se il task ha avuto successo;
2. **P (Partial)**, se il task ha avuto successo, ma con dei problemi riscontrati durante la procedura;
3. **F (Failure)**, se il task è fallito.

I risultati ottenuti sono i seguenti:

Task 1:

Utente	Esito
Maria	S
Luca	S
Francesco	S

Task 2:

Utente	Esito
Maria	P
Luca	S
Francesco	S

Task 3:

Utente	Esito
Maria	P
Luca	S
Francesco	S

Abbiamo assegnato 1 per le task svolte con successo (S), il 0.5 per le task svolte parzialmente(P) e 0 per le task fallite. Il tasso di successo viene calcolato come il rapporto tra la somma del valore assegnato alle task S e il valore assegnato alle task P, e il numero totale di attività eseguite.

Nel nostro caso il tasso di successo è il seguente:

$$T = \frac{7 \cdot 1 + 2 \cdot 0.5}{7 + 2} = \frac{7 + 1}{9} = \frac{8}{9} \approx 0.89\%$$

2.1.7 Glossario

Termine	Descrizione
API	Interfaccia che permette a diverse applicazioni di comunicare tra loro.
Database	Sistema organizzato per la memorizzazione, gestione e recupero di dati.
Framework	Struttura di base su cui un software può essere sviluppato.
Frontend	Parte di un'applicazione software che interagisce direttamente con l'utente.
Backend	Parte di un'applicazione software che gestisce la logica, i database e le operazioni di background.
RESTful API	Stile di architettura per servizi web che utilizza i metodi HTTP e si basa sui principi REST.
UML (Unified Modeling Language)	Linguaggio di modellazione standard per specificare, visualizzare e documentare i componenti di un sistema software.
Use Case Diagram	Diagramma UML che rappresenta le interazioni tra attori e casi d'uso in un sistema.
Sequence Diagram	Diagramma UML che mostra come gli oggetti interagiscono in una sequenza temporale.
Statechart Diagram	Diagramma UML che descrive gli stati di un oggetto e le transizioni tra questi stati.
Personas	Rappresentazioni fittizie di utenti ideali basate su dati di ricerca, utilizzate per guidare le decisioni di design.
Usabilità	Misura in cui un prodotto può essere utilizzato da specifici utenti per raggiungere obiettivi specifici con efficacia, efficienza e soddisfazione.
Android	Sistema operativo mobile sviluppato da Google, utilizzato su una vasta gamma di dispositivi.
Spring Boot	Framework Java che facilita la creazione di applicazioni standalone, basate su Spring, che possono essere eseguite con una configurazione minima.
JPA	Specifiche Java per la gestione della persistenza dei dati tra applicazioni Java e database relazionali.
Testing	Processo di esecuzione di un programma per identificare bug e verificare che il software soddisfi i requisiti specificati.

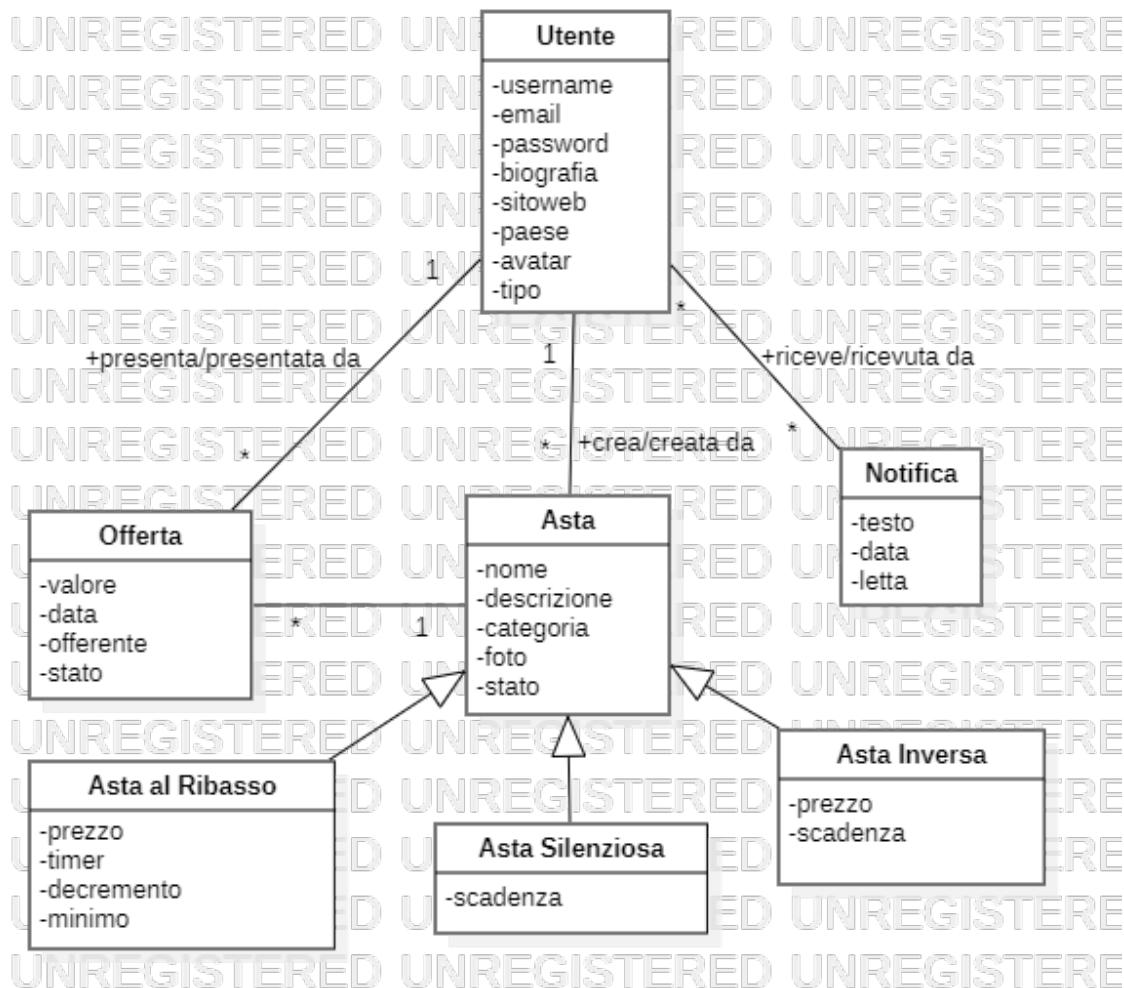
Termine	Descrizione
JUnit	Framework di testing per il linguaggio di programmazione Java.
Client	Dispositivo o programma che accede a servizi forniti da un server.
Server	Dispositivo o programma che fornisce servizi a uno o più client.
HTTP	Protocollo di trasferimento di ipertesti utilizzato per il web.
AWS	Piattaforma di servizi cloud offerta da Amazon, che include una vasta gamma di servizi tra cui calcolo, storage e database.
EC2	Servizio di AWS che fornisce capacità di calcolo scalabile nel cloud.
Docker	Piattaforma per la creazione, il deployment e l'esecuzione di applicazioni in container.
Docker Compose	Strumento per definire e gestire applicazioni multi-container Docker.
PostgreSQL	Sistema di gestione di database relazionale open-source.
Retrofit	Libreria per Android e Java che facilita la comunicazione con servizi web basati su REST.
SonarQube	Piattaforma open-source per l'analisi continua della qualità del codice.
Mock-up	Prototipo non funzionante di un'applicazione, utilizzato per dimostrare e testare il design dell'interfaccia utente.
Container	Unità standardizzata di software che impacchetta il codice e tutte le sue dipendenze, consentendo l'esecuzione dell'applicazione in modo rapido e affidabile da un ambiente all'altro.

2.2 Specifica dei Requisiti

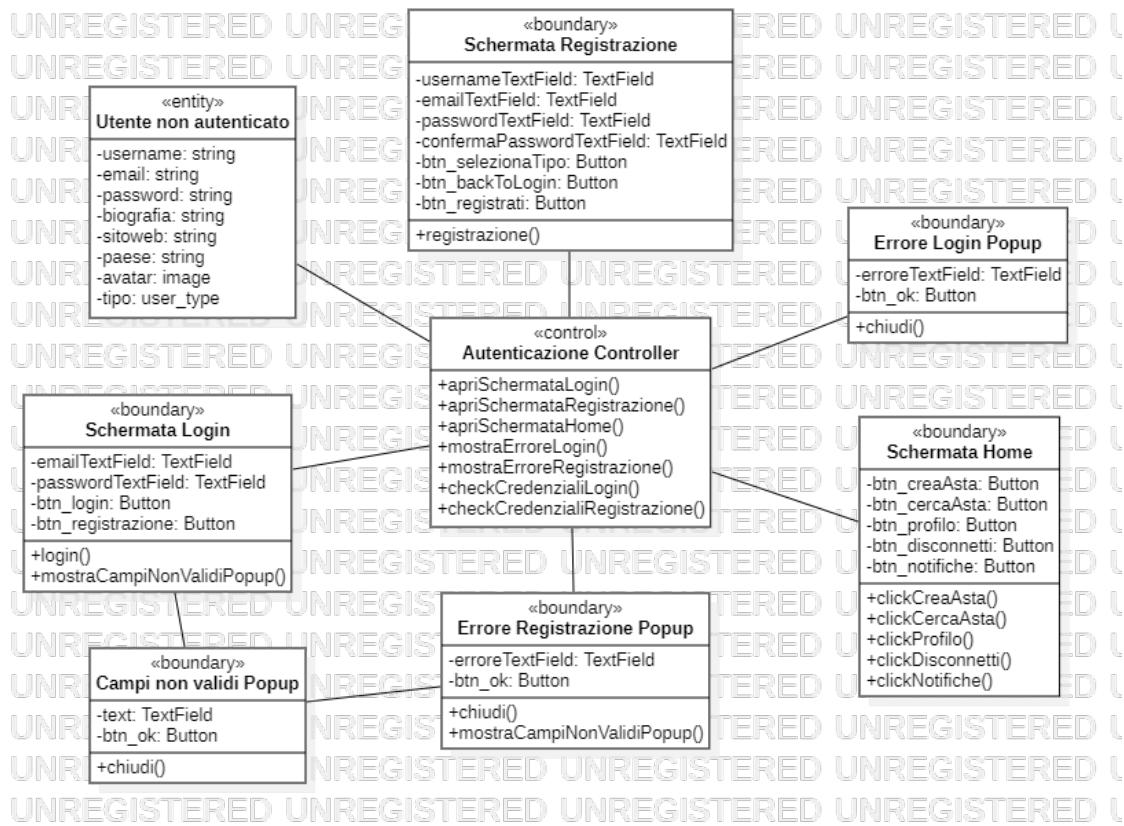
Nella specifica dei requisiti, i requisiti raccolti in fase di analisi vengono documentati dettagliatamente, descrivendo in maniera precisa cosa deve fare il sistema ed in che modo. Di seguito sono mostrati i diagrammi UML sviluppati durante la fase di analisi dei requisiti del progetto.

2.2.1 Classi, oggetti e relazioni di analisi

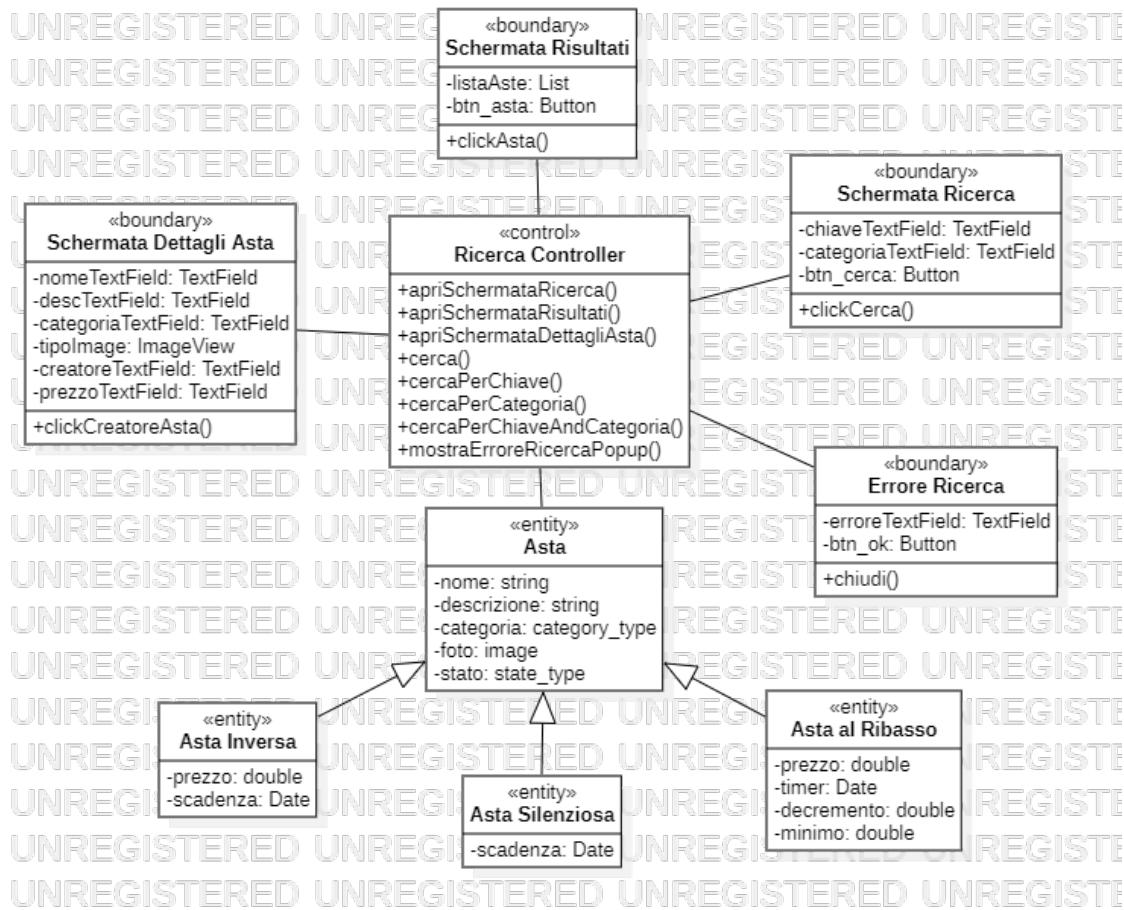
Class Diagram Entità



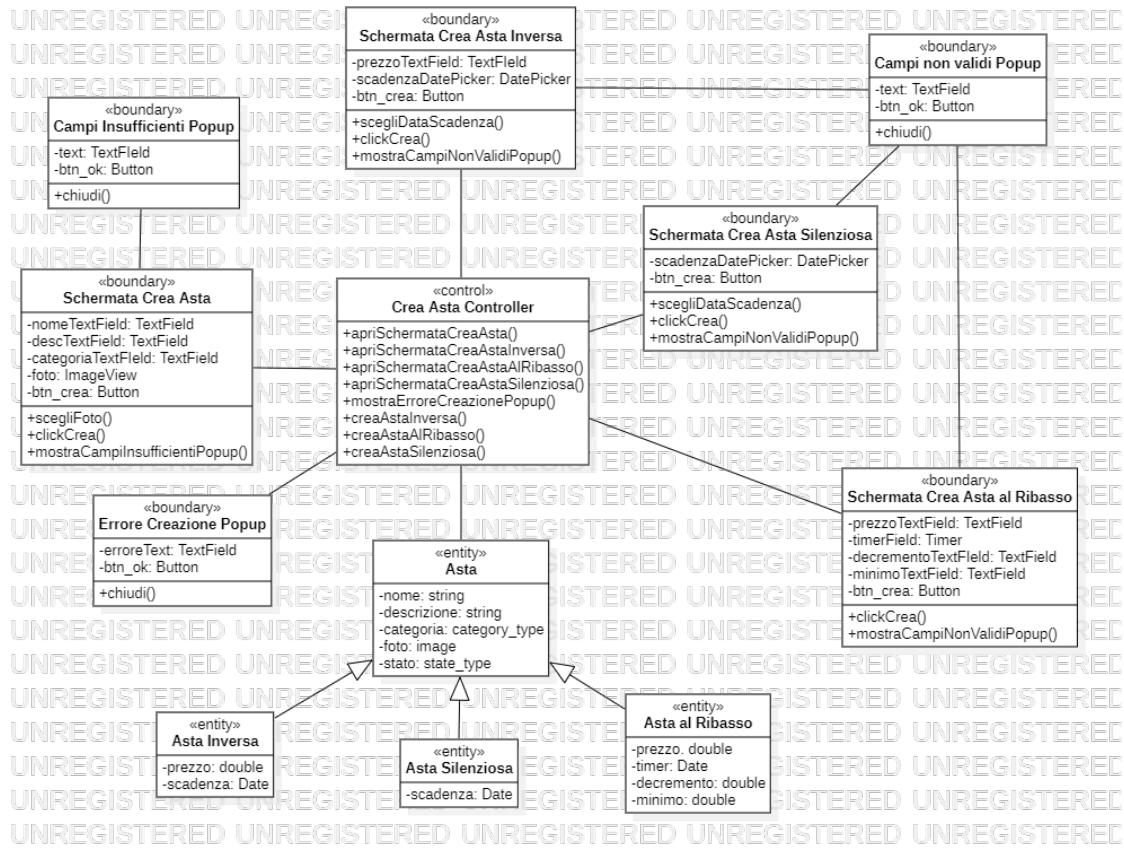
Class Diagram Autenticazione



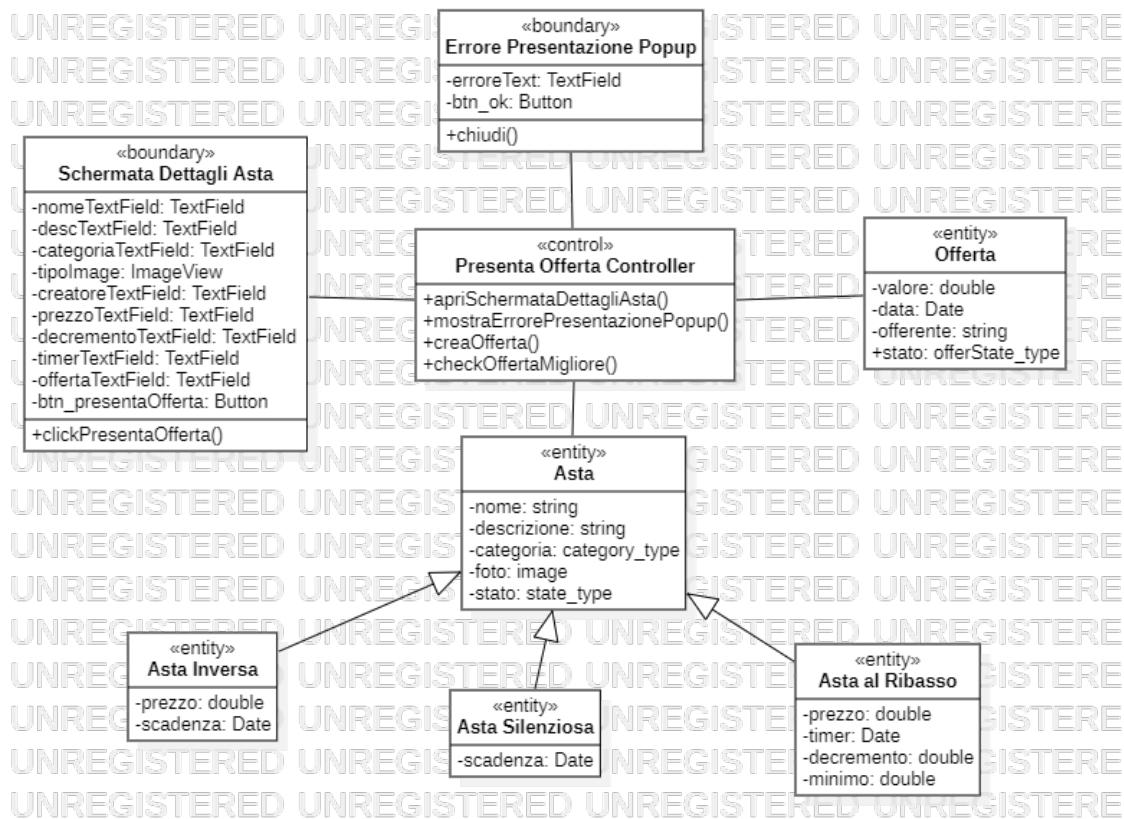
Class Diagram Ricerca Asta



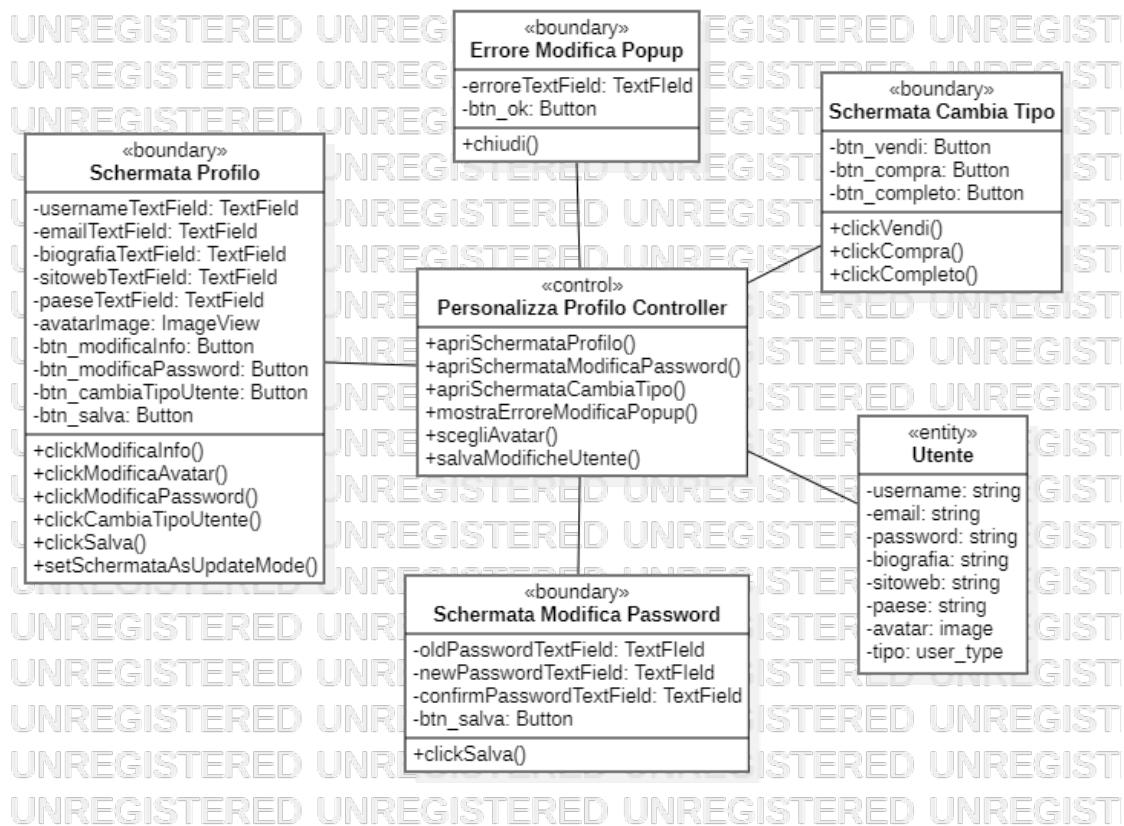
Class Diagram Creazione Asta



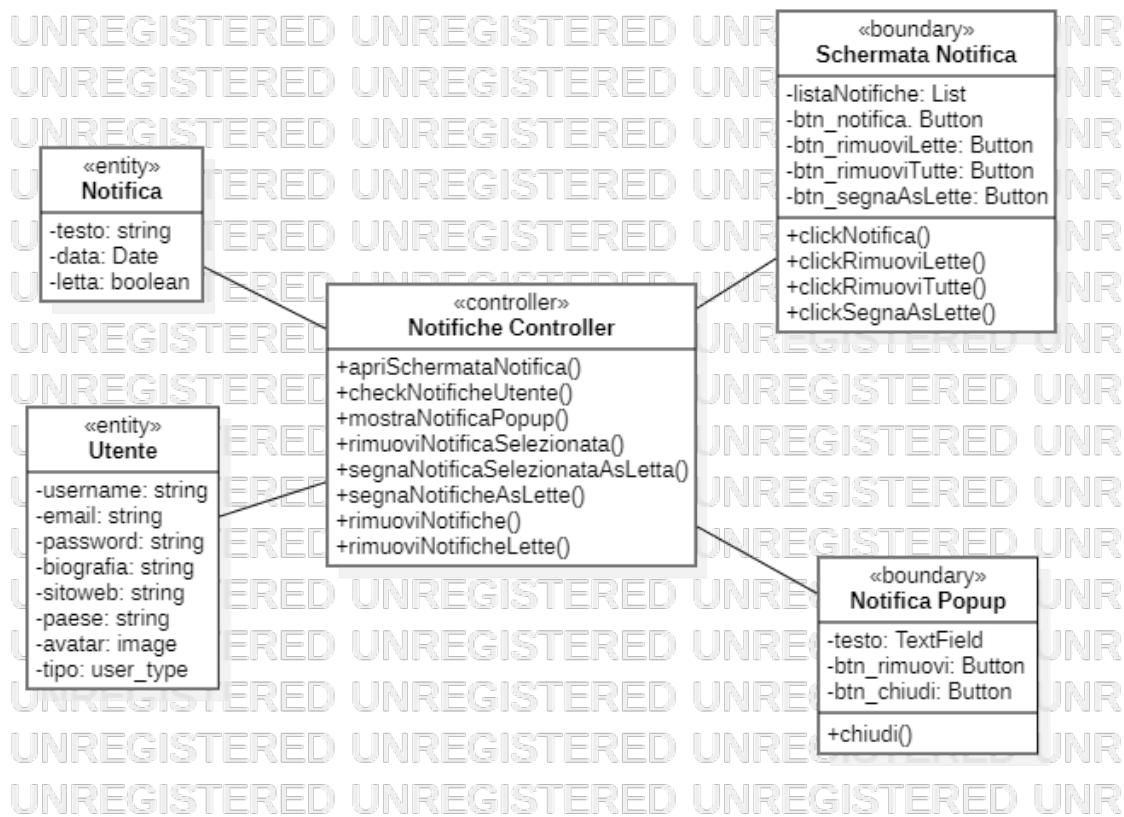
Class Diagram Presentazione Offerta



Class Diagram Personalizzazione Profilo



Class Diagram Gestione Notifiche

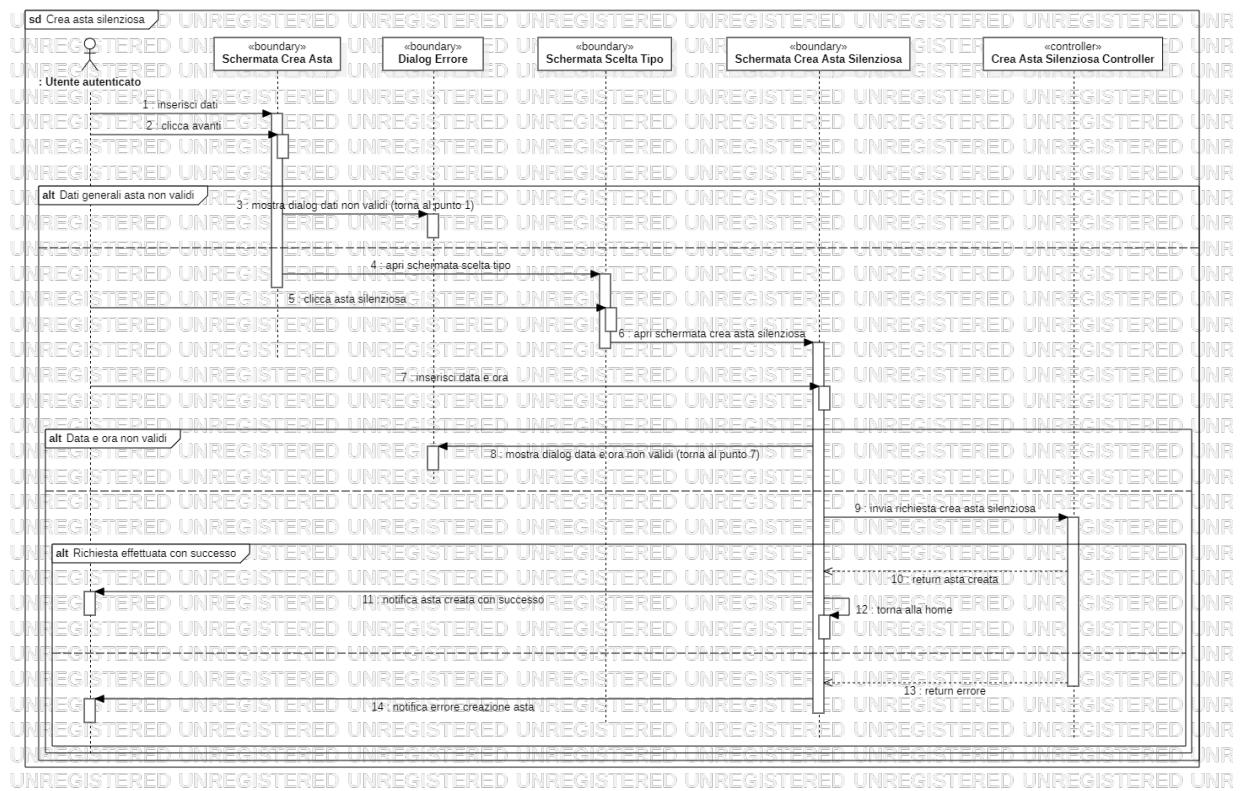


2.2.2 Diagrammi di sequenza di analisi

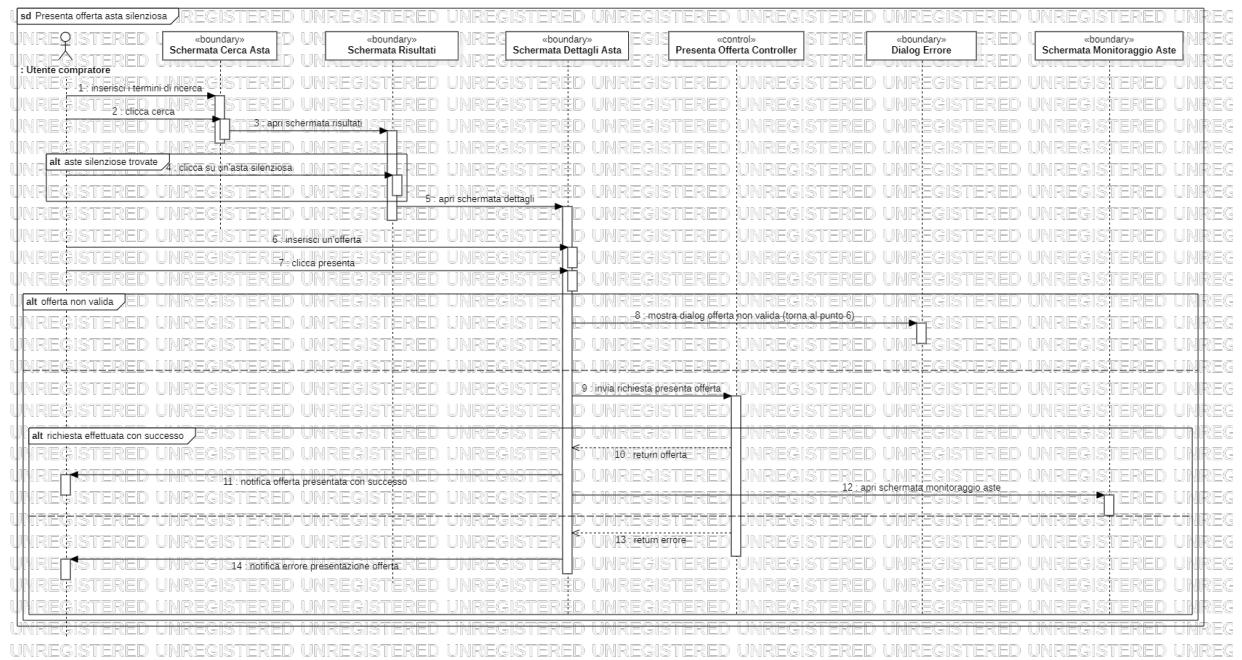
Di seguito sono elencati i sequence diagram, i quali mostrano le interazioni tra le varie componenti software del sistema e l'ordine in cui avvengono tali operazioni, ideati in fase di analisi. In particolare, i casi d'uso presi in considerazione sono:

1. Creazione di un'asta silenziosa
2. Presentazione di un'offerta per un'asta silenziosa

Sequence Diagram Crea Asta Silenziosa



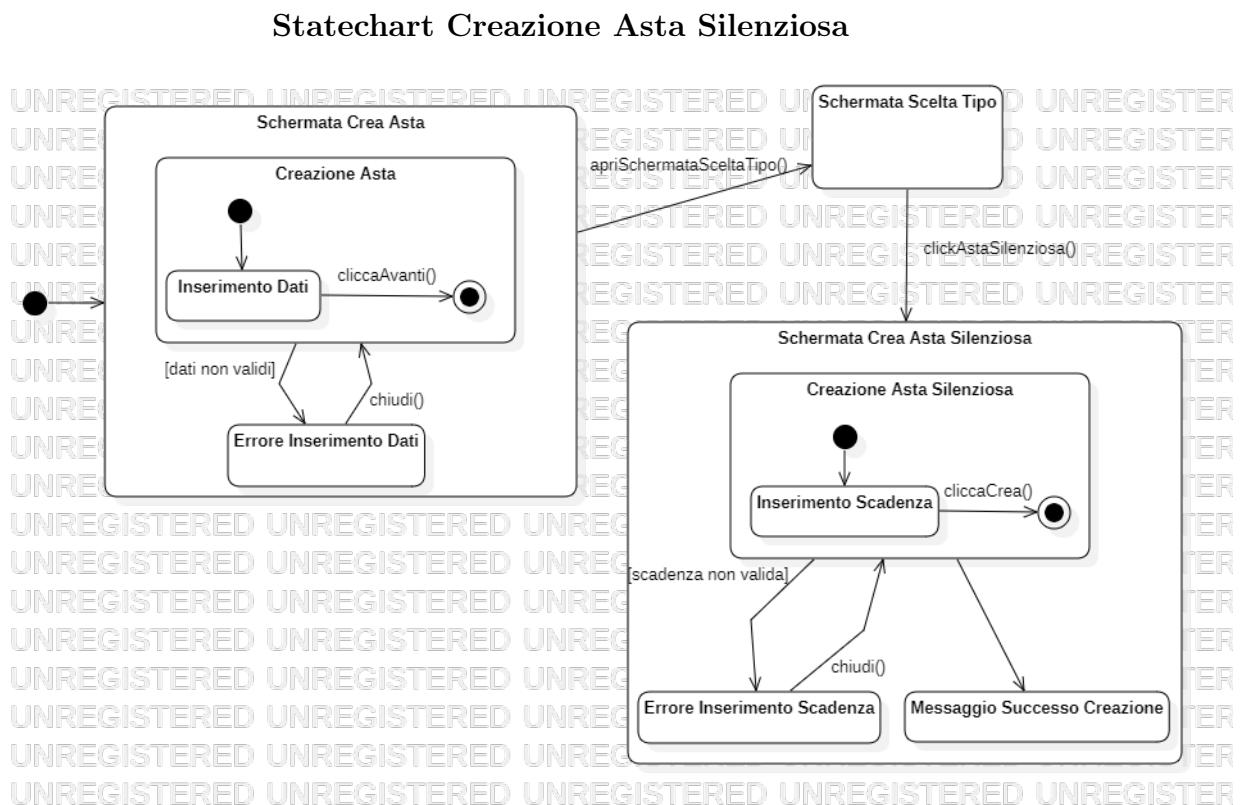
Sequence Diagram Presenta Offerta Asta Silenziosa



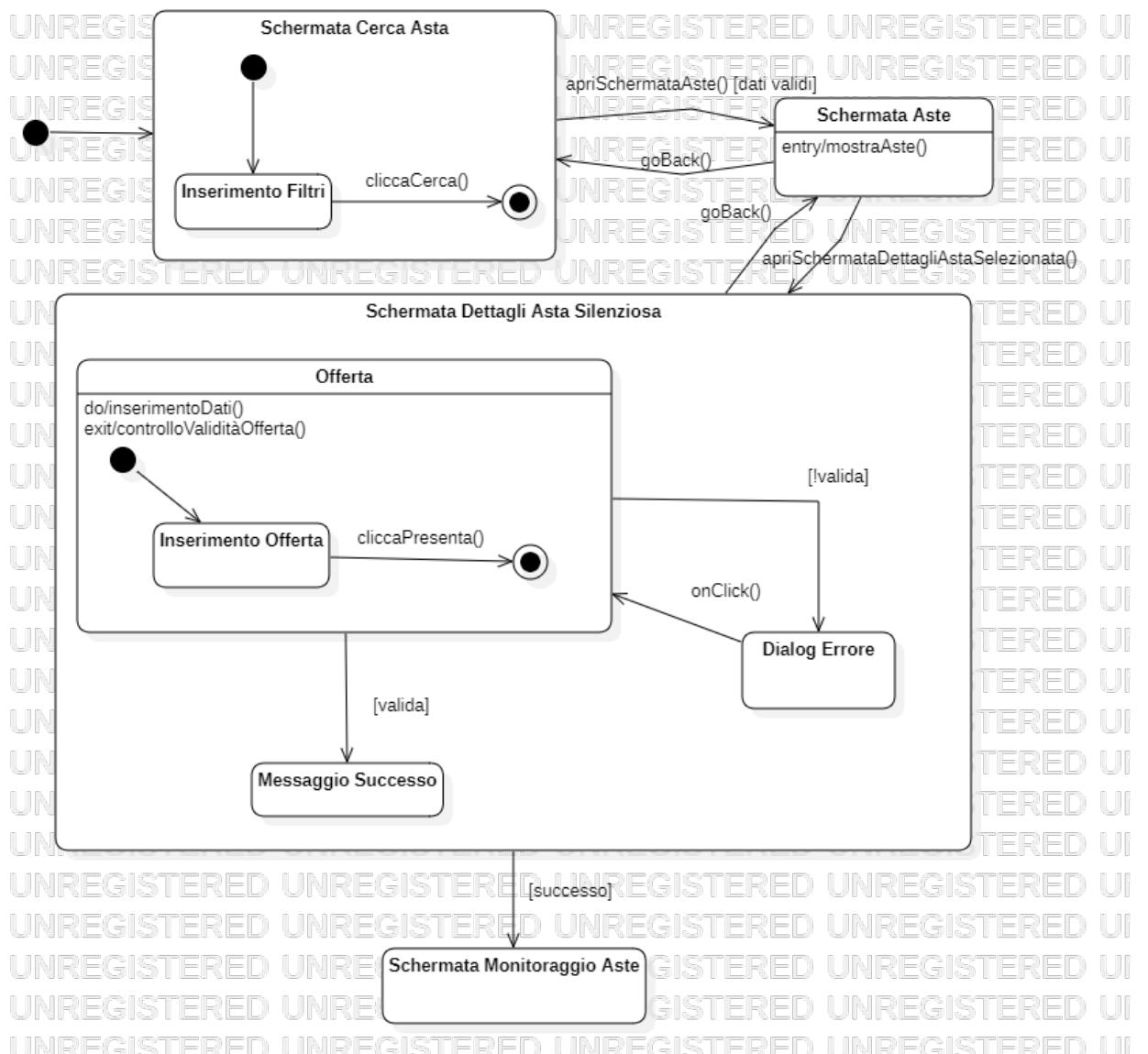
2.2.3 Statecharts dell'interfaccia grafica

In questa sezione vengono mostrati gli statechart dell'interfaccia grafica, i quali rappresentano gli stati possibili di un oggetto e le transizioni tra questi stati in risposta a eventi o condizioni. Abbiamo creato 4 diagrammi di stato per i seguenti casi d'uso:

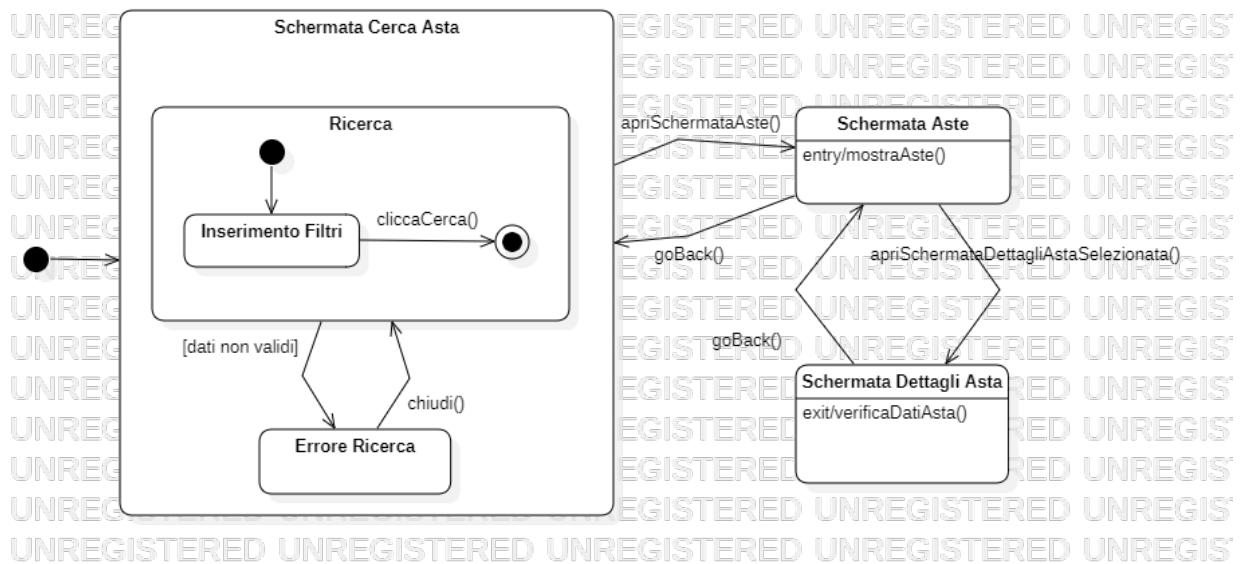
1. Creazione di un'asta silenziosa
2. Presentazione di un'offerta per un'asta silenziosa
3. Ricerca asta
4. Visualizzazione profilo creatore asta



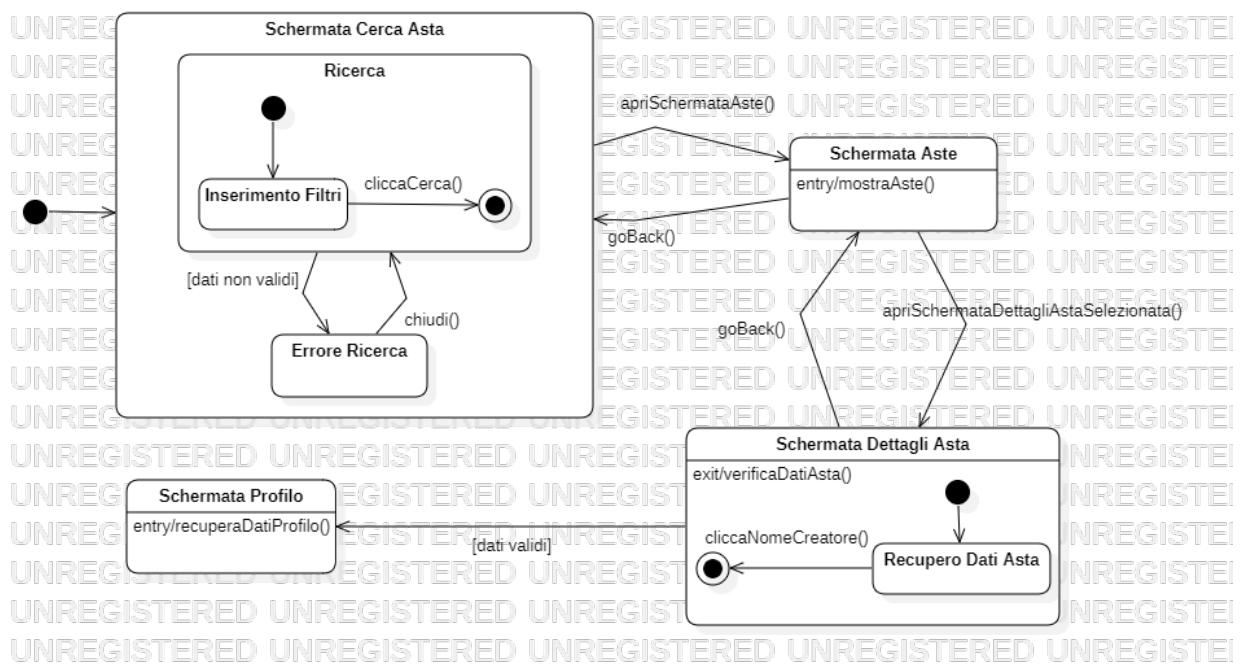
Statechart Presenta Offerta Asta Silenziosa



Statechart Ricerca Asta



Statechart Visualizzazione Profilo Creatore Asta



3 Design del sistema

In questo capitolo verrà fornita una descrizione dettagliata di come il sistema è progettato e strutturato. Il design del sistema è una parte essenziale perché guida gli sviluppatori, i manutentori e gli stakeholder a comprendere l'architettura, i componenti principali, le loro interazioni e il razionale dietro le scelte progettuali fatte.

3.1 Descrizione dell'architettura del sistema

Abbiamo sviluppato il sistema considerando i seguenti componenti architetturali:

1. **Frontend** (Android);
2. **Backend** (Spring Boot);
3. **Database** (Postgresql).

Il Backend e Database sono stati inseriti in container **Docker** ed avviati mediante un docker compose che crea l'infrastruttura del sistema. La comunicazione tra Client e Backend avviene mediante servizi RESTful e **API Rest**, con l'uso della libreria **Retrofit** per la gestione delle richieste HTTP. Il backend, invece, recupera i dati all'interno del database utilizzando i servizi offerti da **JPA Hibernate**. L'intera infrastruttura è ospitata su un'istanza di **Amazon EC2**.

3.1.1 Sviluppo Frontend

La parte frontend del sistema è stata sviluppata mediante il linguaggio di programmazione Java, con l'ambiente di sviluppo **Android Studio**. La scelta di tale ambiente è motivata dalla possibilità di usufruire di varie funzionalità; troviamo tra le principali:

1. Supporto per la compilazione basata su gradle.
2. Refactoring specifico per Android.
3. Un ampio editor di layout e la possibilità di visualizzare in anteprima il layout su più configurazioni dello schermo.
4. Procedure guidate basate su modelli per creare progetti e componenti Android comuni
5. Dispositivi virtuali dove installare ed avviare le proprie applicazioni.
6. Integrazione con sistemi di Versioning.

Android studio, inoltre, è l'ambiente di sviluppo ufficiale di Google per Android, il che significa che è ottimizzato e aggiornato regolarmente con le ultime funzionalità e API di Android.

3.1.2 Sviluppo Backend

La parte backend è stata sviluppato tramite il linguaggio di programmazione Java, sull'ambiente di sviluppo Eclipse, con l'utilizzo del framework **Spring Boot**. Spring Boot è un potente strumento che consente agli oggetti di definire le proprie dipendenze che successivamente il container Spring provvederà ad inserire. Tra i vari vantaggi che si hanno utilizzando questa tecnologia, troviamo:

1. Riduzione della configurazione necessaria per avviare un progetto, grazie alle convenzioni predefinite e alle configurazioni automatiche.
2. Creazione di applicazioni standalone, pronte per la produzione, con un semplice file JAR o WAR, che accelerano il ciclo di sviluppo.
3. Inclusione di server web embedded come Tomcat, Jetty e Undertow, A differenza di Spring, quindi, non c'è la necessità di configurare separatamente un server esterno.
4. Integrazione con Spring Security, che fornisce meccanismi avanzati per l'autenticazione e l'autorizzazione, semplificando la protezione delle applicazioni.
5. Alta scalabilità: Spring Boot consente di gestire facilmente un aumento del carico e della complessità dell'applicazione.

3.1.3 API Rest

Una API REST (Representational State Transfer) è un'interfaccia che consente ai sistemi di comunicare tra loro utilizzando il protocollo HTTP. Le API RESTful sono progettate per sfruttare le operazioni standard del protocollo HTTP come GET, POST, PUT, DELETE, ecc. Queste operazioni corrispondono alle operazioni CRUD (Create, Read, Update, Delete) sui dati. Di seguito sono elencate le API Rest implementate per lo sviluppo del software DietiDeals24:

Metodo HTTP	URL del Metodo	Descrizione
POST	/auth/login	Esegue il login di un utente
POST	/auth/google/	Accede al sistema mediante credenziali Google

Table 6: API Rest per Autenticazione

Metodo HTTP	URL del Metodo	Descrizione
POST	/utente/registra	Registra un nuovo utente
GET	/utente/recupera/id	Recupera un utente specificato dall'ID
POST	/utente/aggiorna	Aggiorna i dati di un utente
POST	/utente/modPassword	Modifica la password di un utente

Table 7: API Rest per Utente

Metodo HTTP	URL del Metodo	Descrizione
POST	/offerta/crea	Crea una nuova offerta
GET	/offerta/recuperaOrdinate/id_asta	Recupera le offerte ordinate per un'asta specificata
GET	/offerta/recuperaPerUtente/id_utente	Recupera le offerte fatte da un utente specificato
PUT	/offerta/accetta/id_offerta	Accetta un'offerta specificata dall'ID
PUT	/offerta/rifiuta/id_offerta	Rifiuta un'offerta specificata dall'ID

Table 8: API Rest per Offerta

Metodo HTTP	URL del Metodo	Descrizione
GET	/asta/cercaTutte	Ottiene tutte le aste
GET	/asta/cercaPerUtente/id_creatore	Ottiene le aste create da un utente specificato
GET	/asta/cercaPerChiave/chiave	Cerca le aste per parola chiave
GET	/asta/cercaPerCategoria/categoria	Cerca le aste per categoria
GET	/asta/cercaPerChiaveAndCategoria/chiave&categoria	Cerca le aste per parola chiave e categoria
GET	/asta/cercaPerOfferteUtente/id_utente	Ottiene le aste in cui un utente ha fatto offerte
POST	/asta/creaAstaInversa	Crea una nuova asta inversa
POST	/asta/creaAstaAlRibasso	Crea una nuova asta al ribasso
POST	/asta/creaAstaSilenziosa	Crea una nuova asta silenziosa
GET	/asta/dettagliAstaInversa/id	Ottiene i dettagli di un'asta inversa specificata dall'ID
GET	/asta/dettagliAstaRibasso/id	Ottiene i dettagli di un'asta al ribasso specificata dall'ID
GET	/asta/dettagliAstaSilenziosa/id	Ottiene i dettagli di un'asta silenziosa specificata dall'ID

Table 9: API Rest per Asta

Metodo HTTP	URL del Metodo	Descrizione
GET	/notifica/mostraTutte/id_utente	Mostra tutte le notifiche di un utente
POST	/notifica/rimuovi/id	Rimuove una notifica specificata dall'ID
POST	/notifica/rimuoviLette/id_utente	Rimuove tutte le notifiche lette di un utente
POST	/notifica/svuota/id_utente	Svuota tutte le notifiche di un utente
PUT	/notifica/segna/id	Segna una notifica come letta specificata dall'ID
PUT	/notifica/segnaTutte/id_utente	Segna tutte le notifiche di un utente come lette

Table 10: API Rest per Notifica

3.1.4 PostgreSQL

Il DBMS usato per lo storage e management dei dati della piattaforma è PostgreSQL, basato sul modello relazionale. Abbiamo deciso di utilizzare un DMBS relazionale poichè questo modello ci è sembrato adeguato alle esigenze richieste dal sistema.

3.1.5 JPA Hibernate

JPA (Java Persistence API) e Hibernate sono tecnologie per la gestione della persistenza dei dati all'interno di applicazioni Java. JPA è una specifica standard che definisce un'API per la gestione dei dati relazionali in Java, consentendo agli sviluppatori di interagire con i database utilizzando oggetti Java anziché SQL. Abbiamo scelto Hibernate, una delle implementazioni più diffuse di JPA, poichè offre una serie di funzionalità avanzate per il mapping degli oggetti al database, gestione delle sessioni, e query ad alte prestazioni. Hibernate semplifica notevolmente la gestione della persistenza, riducendo la quantità di codice e migliorando la manutenibilità dell'applicazione.

3.1.6 Retrofit

Retrofit è una libreria open-source per Android e Java che facilita le chiamate HTTP. Utilizza annotazioni per definire le interfacce di comunicazione con i servizi web, semplificando l'integrazione delle API RESTful nelle applicazioni. Con Retrofit, le risposte delle API possono essere automaticamente convertite in oggetti Java tramite converter come, nel nostro caso, Gson. La scelta di usare questa libreria nel sistema Dietideals24 è risultata ottimale poichè le richieste create da Retrofit si integrano perfettamente con un framework come SpringBoot.

3.1.7 Docker

Docker è una piattaforma open-source che automatizza la distribuzione di applicazioni all'interno di container, garantendo che funzionino sempre nello stesso modo ed in maniera regolare. I container Docker includono tutto il necessario per eseguire un'applicazione, comprese librerie, dipendenze e configurazioni, isolandole dal sistema operativo host. L'uso di Docker ha consentito maggiore portabilità al sistema, migliorando l'efficienza nell'uso delle risorse.

L'infrastruttura server viene creata tramite un file chiamato '**docker-compose.yml**' che ci permette di definire ed esporre i container che faranno parte del sistema. Nel nostro caso all'interno del file abbiamo definito il Backend ed il Database.

3.1.8 Amazon EC2

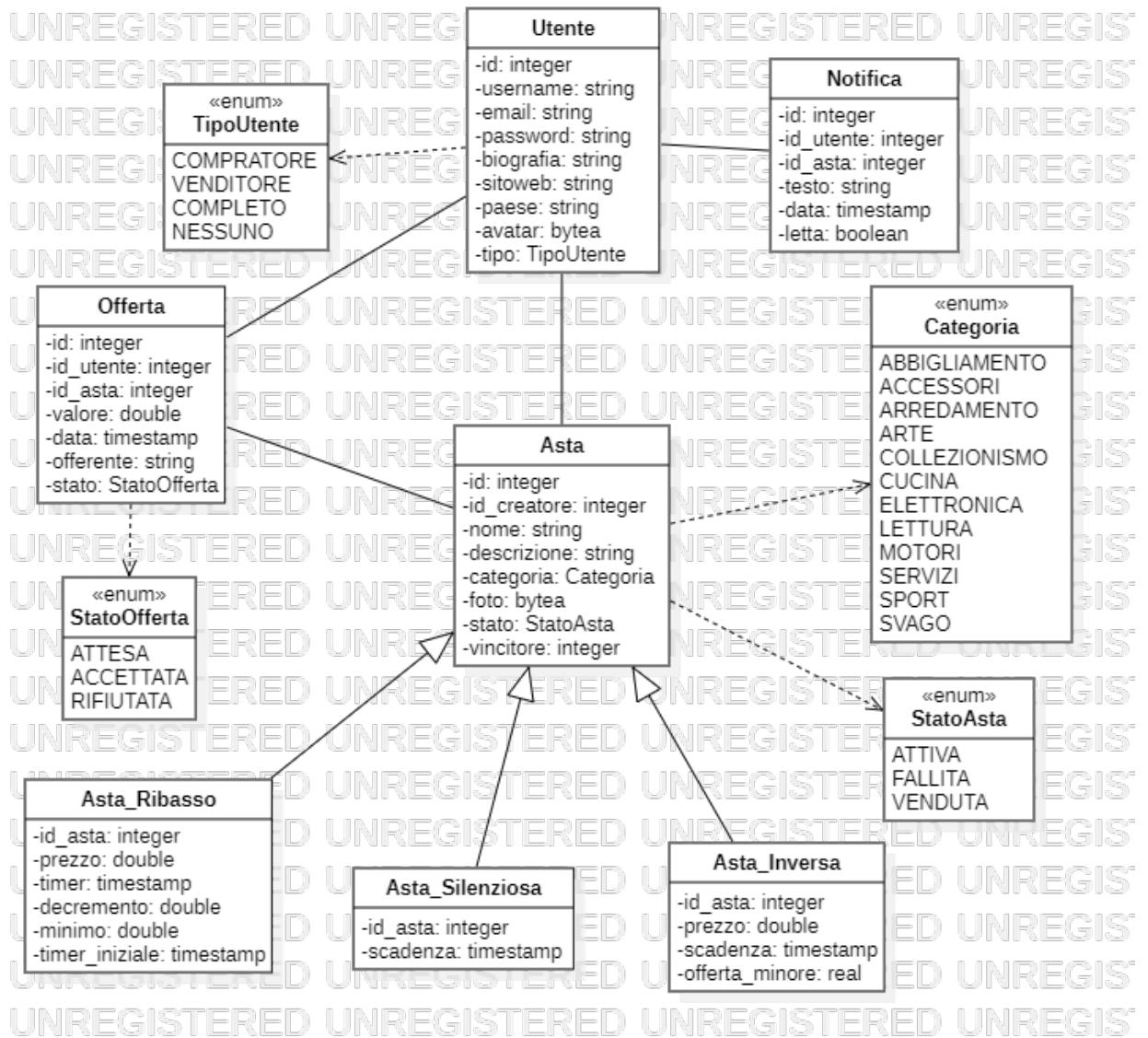
L'applicazione è stata rilasciata su Amazon EC2, un'istanza Cloud che fornisce un ambiente Linux su cui lanciare i Container. Ciò ci ha permesso di poter esporre il software su un indirizzo pubblico, creando il server dell'applicativo mediante docker-compose all'interno dell'istanza.

The screenshot shows the AWS EC2 Instances page. At the top, there are tabs for 'Istanze (1/1)' and 'Informazioni'. To the right, it says 'Last updated less than a minute ago' and has buttons for 'Connetti', 'Stato dell'istanza', and 'Operazioni'. Below this is a search bar with placeholder text 'Trova Istanza per attributo o tag (case-sensitive)' and a dropdown for 'Tutti gli stati'. There are filters for 'Name' (with a pencil icon), 'ID istanza', 'Stato dell'istanza' (with a dropdown arrow), 'Tipo di istanza' (with a dropdown arrow), and 'Verifica dello stato'. A single instance is listed: 'Server Dietideals24' (ID: i-0f9979d965a035727). It is shown as 'In esecuzione' (Running) with a green checkmark, 't2.micro' as its instance type, and '2/2 controlli superati' (2/2 controls passed) with a green checkmark. The bottom of the page shows the instance ID: 'i-0f9979d965a035727 (Server Dietideals24)'.

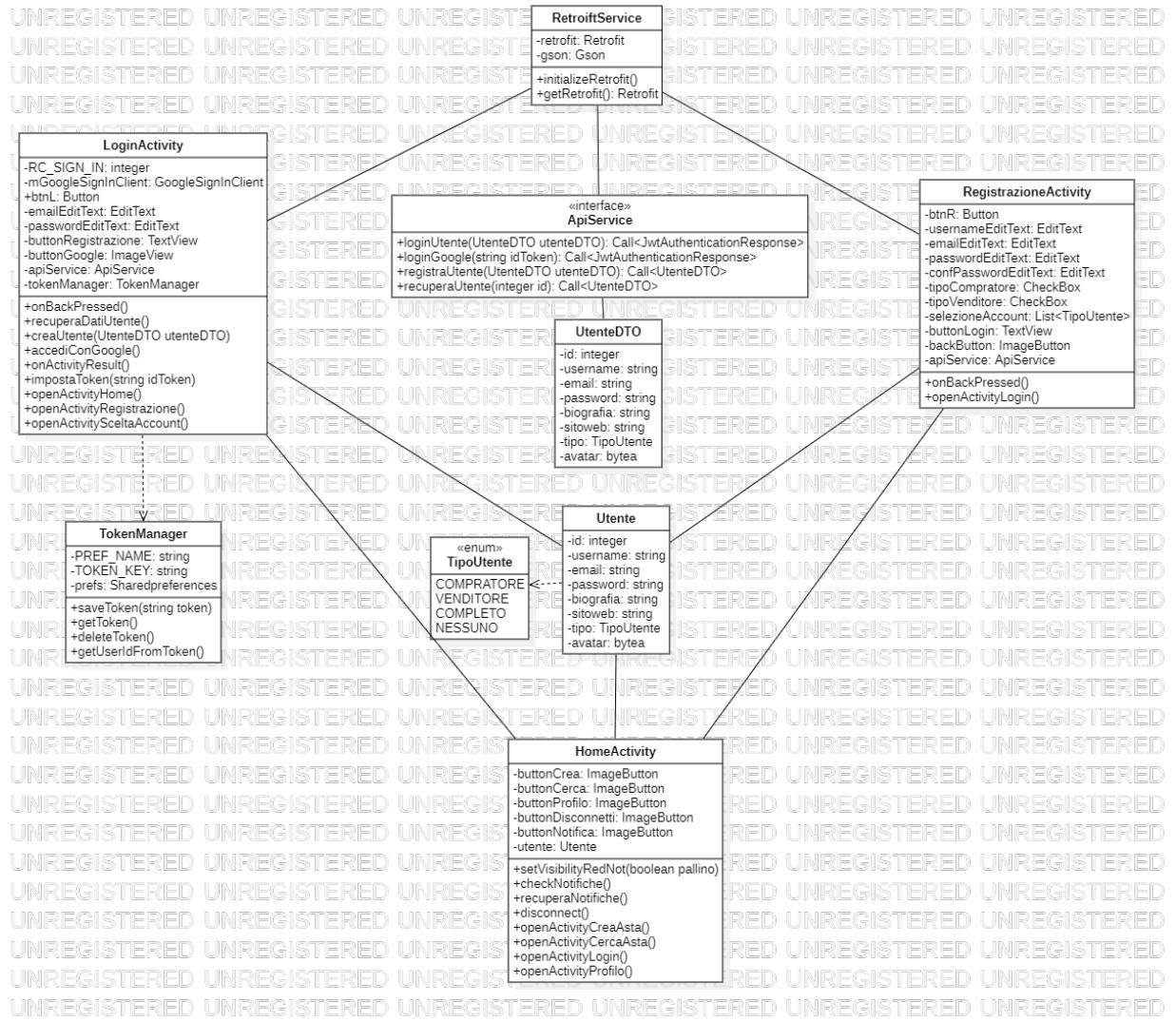
3.2 Diagrammi di classi di design

In questa sezione sono elencati i Class diagram UML che descrivono graficamente le funzionalità del sistema ed il modo in cui sono state implementate.

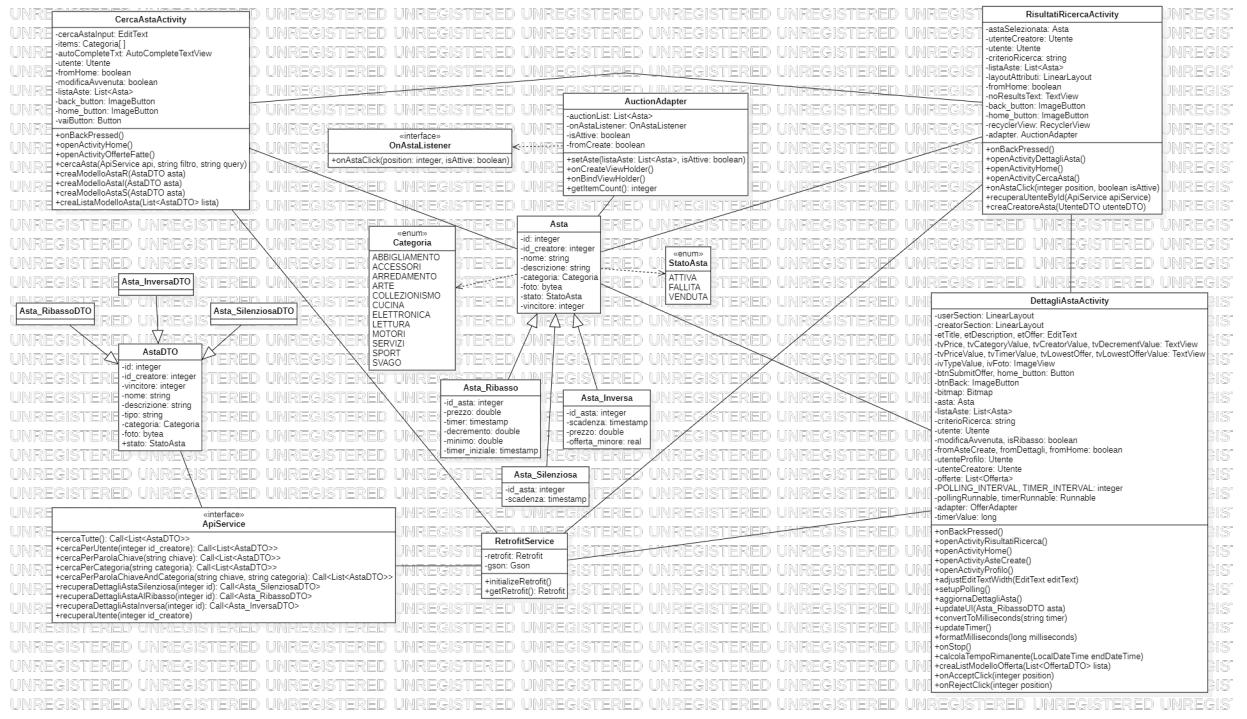
3.2.1 Entità



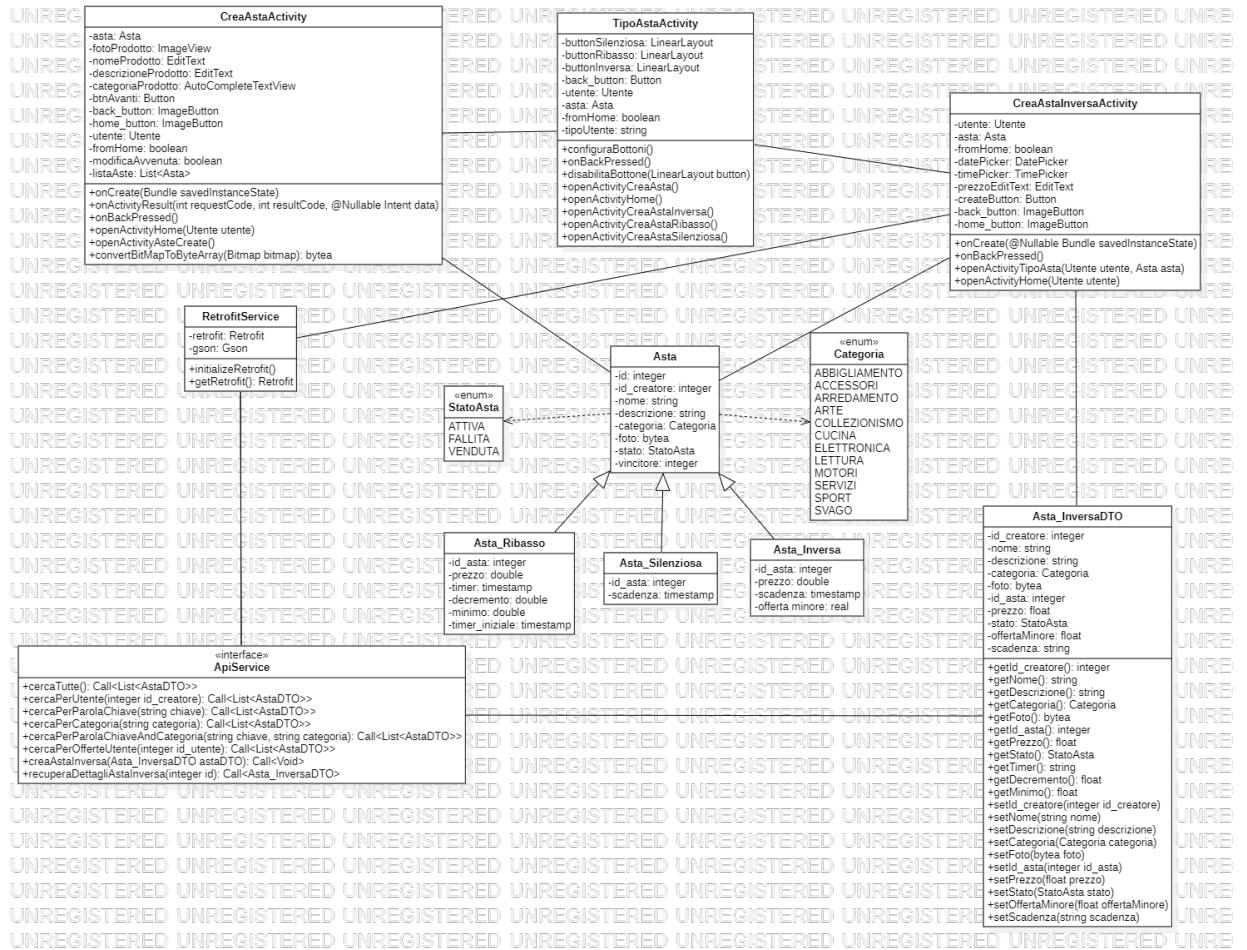
3.2.2 Autenticazione



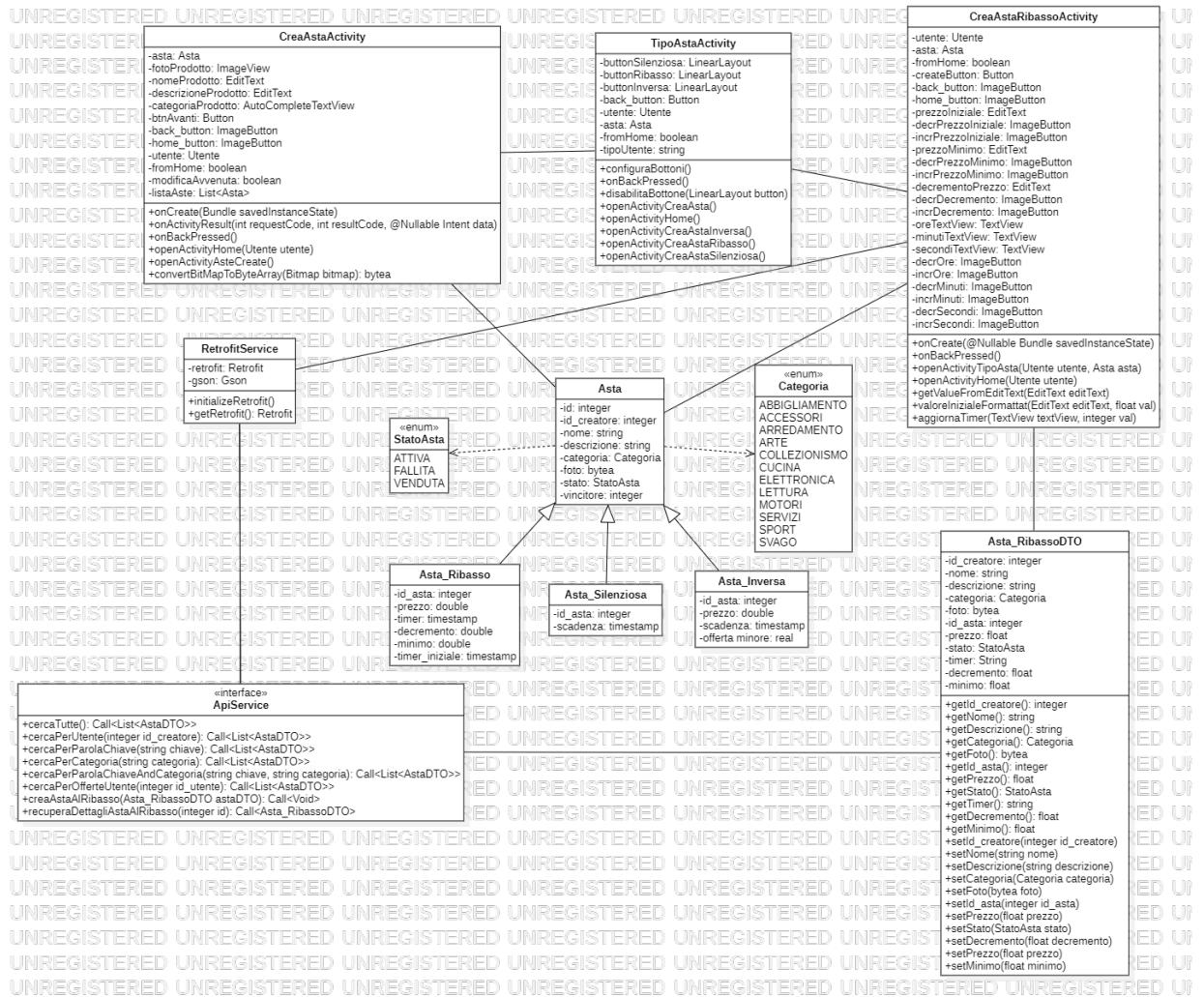
3.2.3 Ricerca Asta



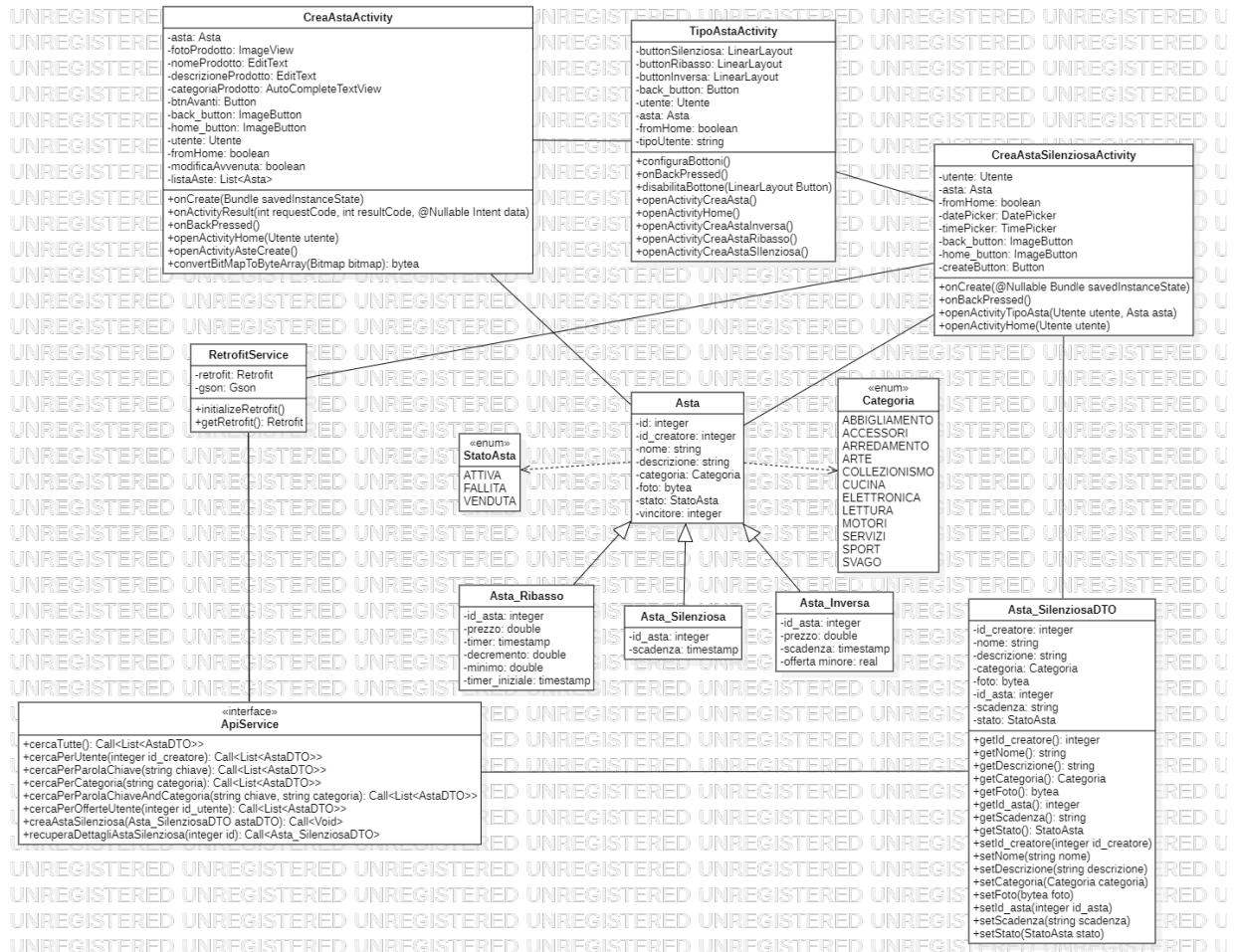
3.2.4 Creazione Asta Inversa



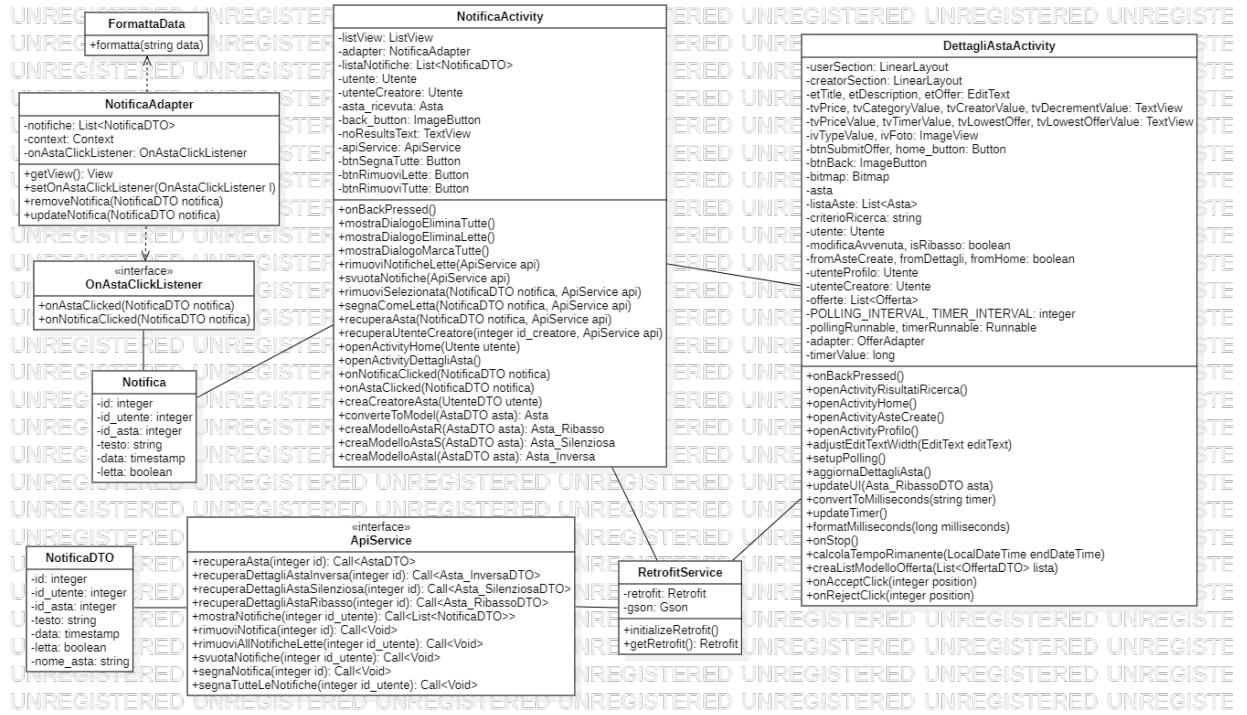
3.2.5 Creazione Asta al Ribasso



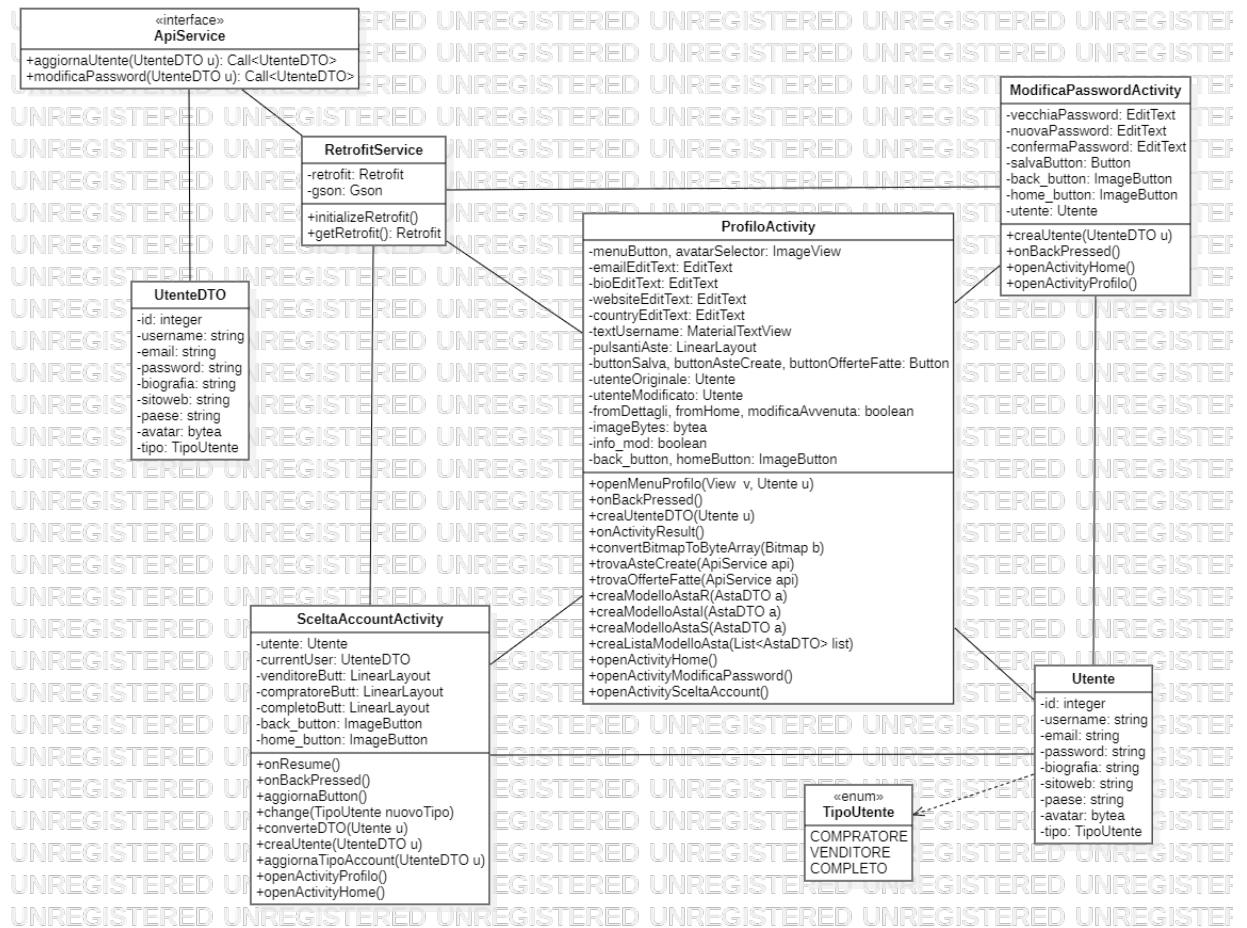
3.2.6 Creazione Asta Silenziosa



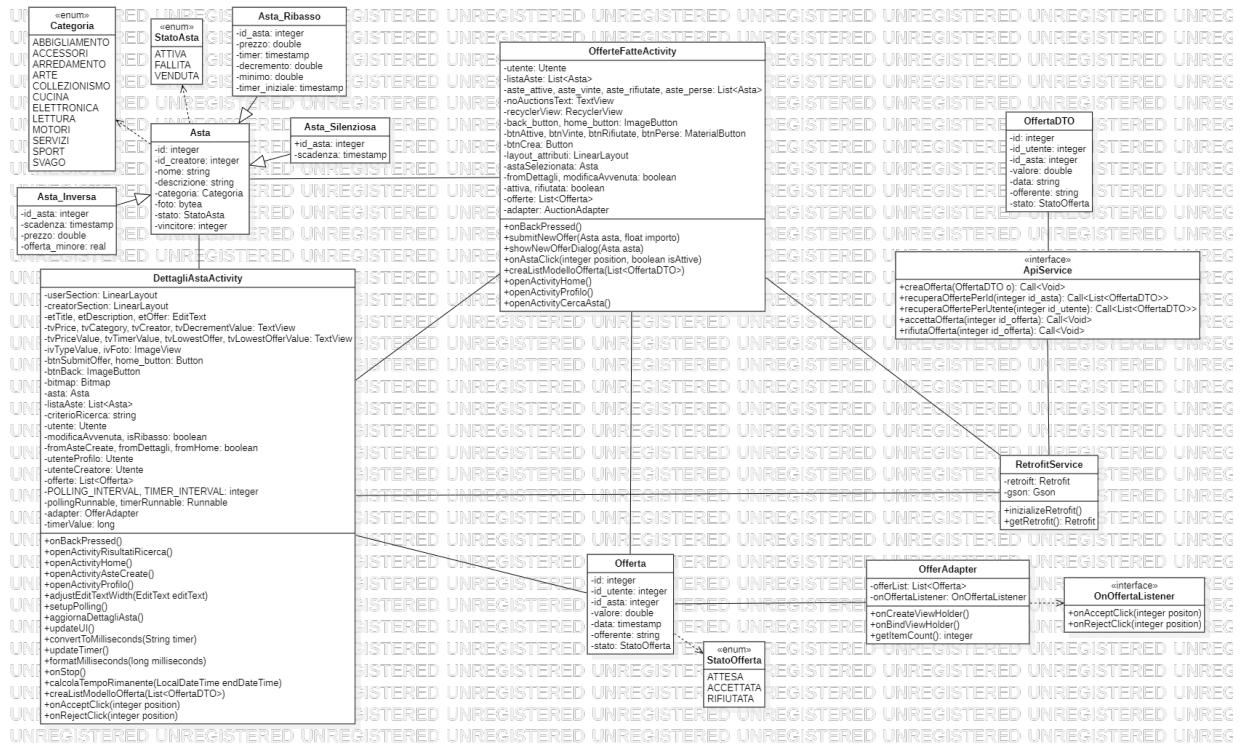
3.2.7 Gestione Notifiche



3.2.8 Personalizzazione Profilo



3.2.9 Presentazione Offerta

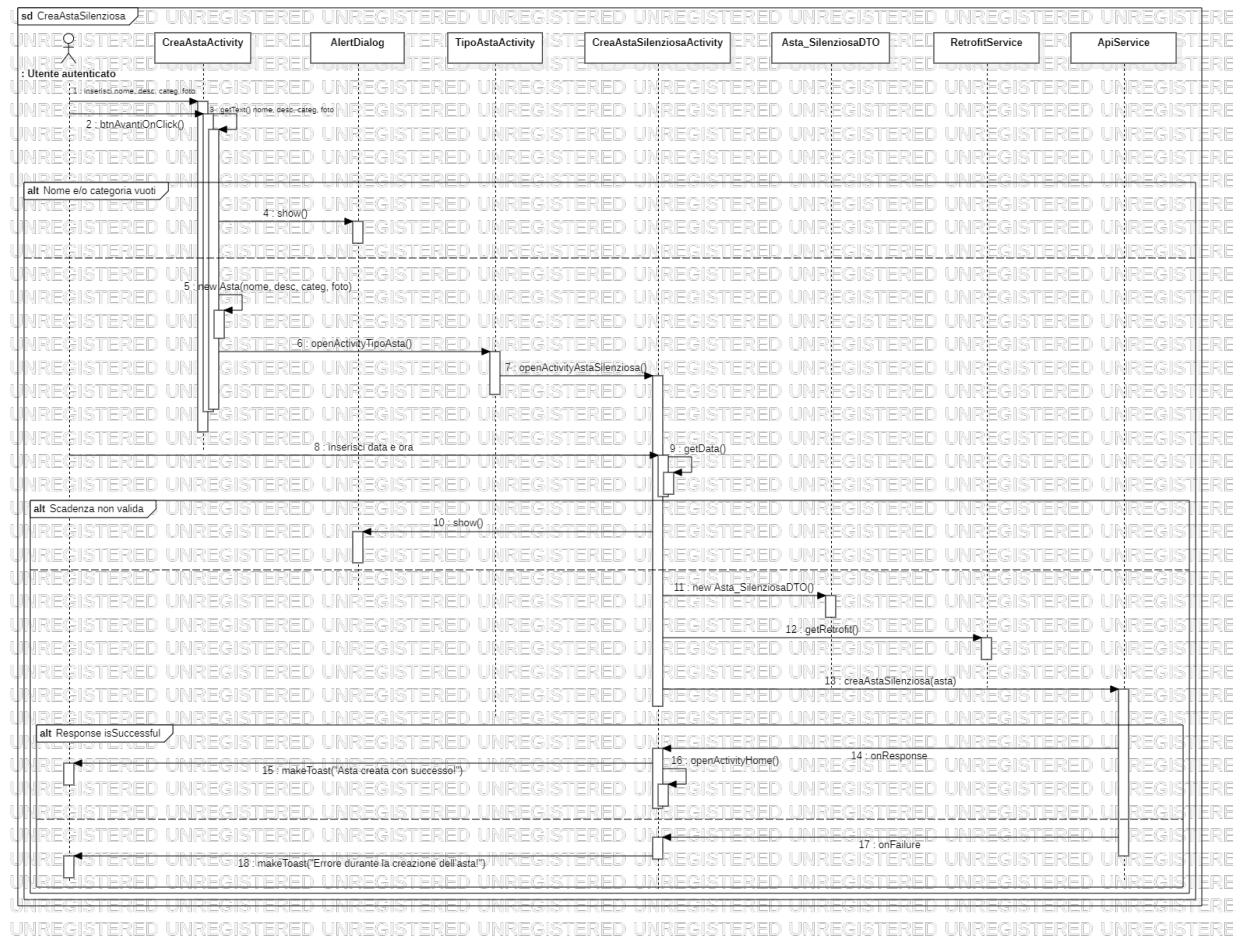


3.3 Diagrammi di sequenza di design

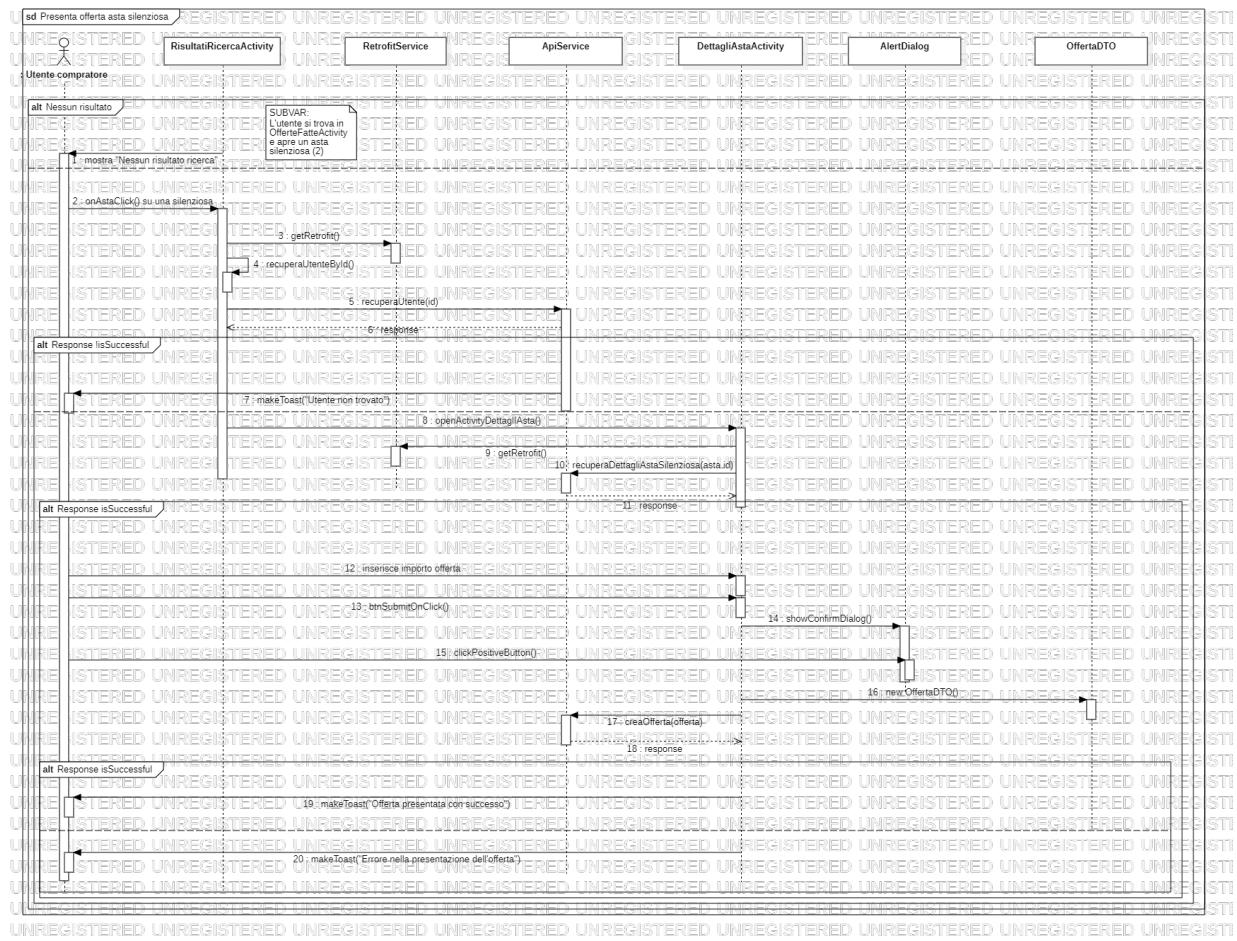
In questa sezione verranno elencati i sequence diagram di design del sistema, i quali mostrano i passi e le interazioni tra le varie componenti per eseguire una specifica funzionalità implementata nel software. In particolare, abbiamo preso in considerazione i seguenti casi d'uso:

- 1. Creazione asta silenziosa**
- 2. Presentazione offerta per asta silenziosa**
- 3. Ricerca asta**
- 4. Visualizzazione profilo creatore asta**

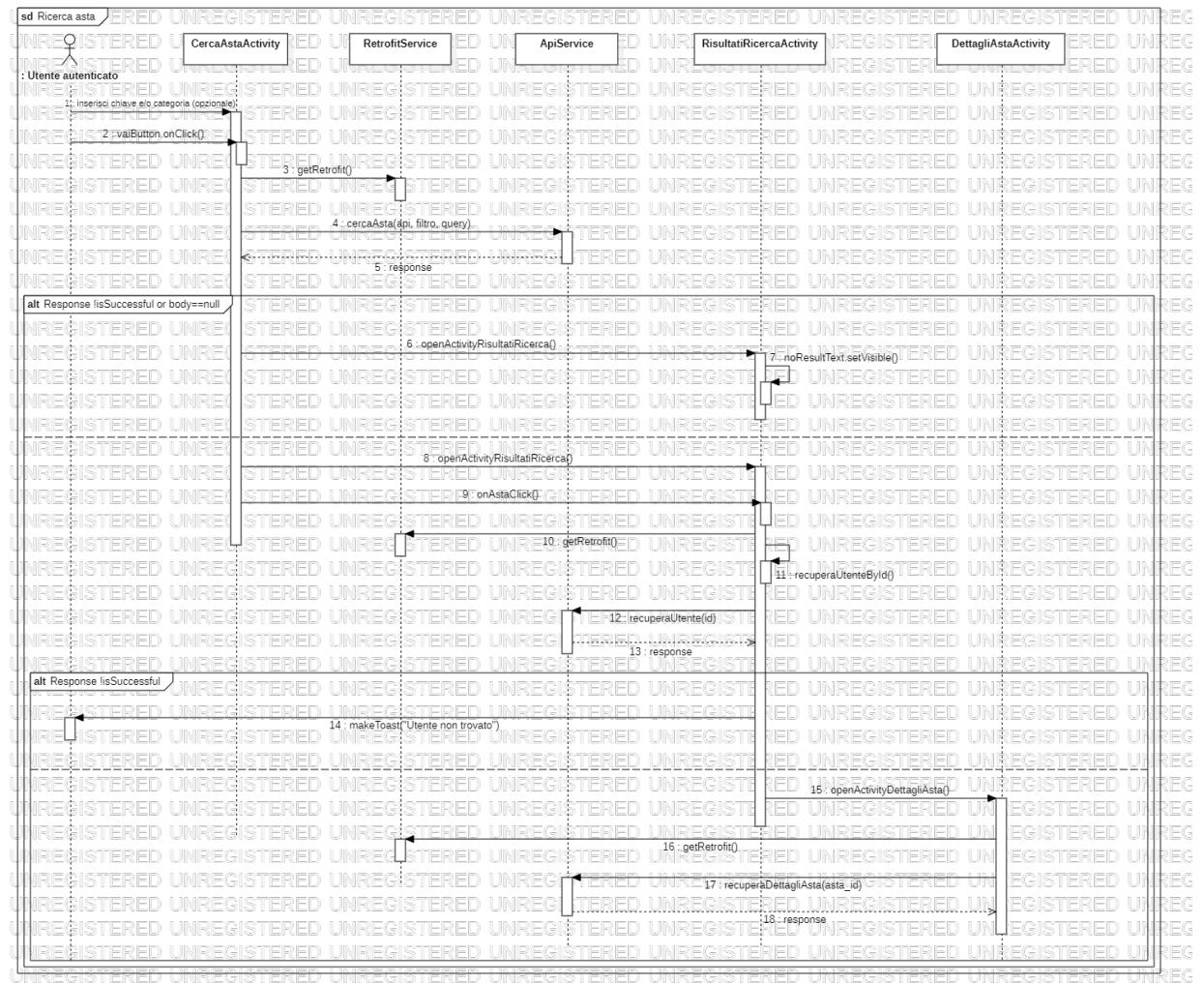
3.3.1 Creazione Asta Silenziosa



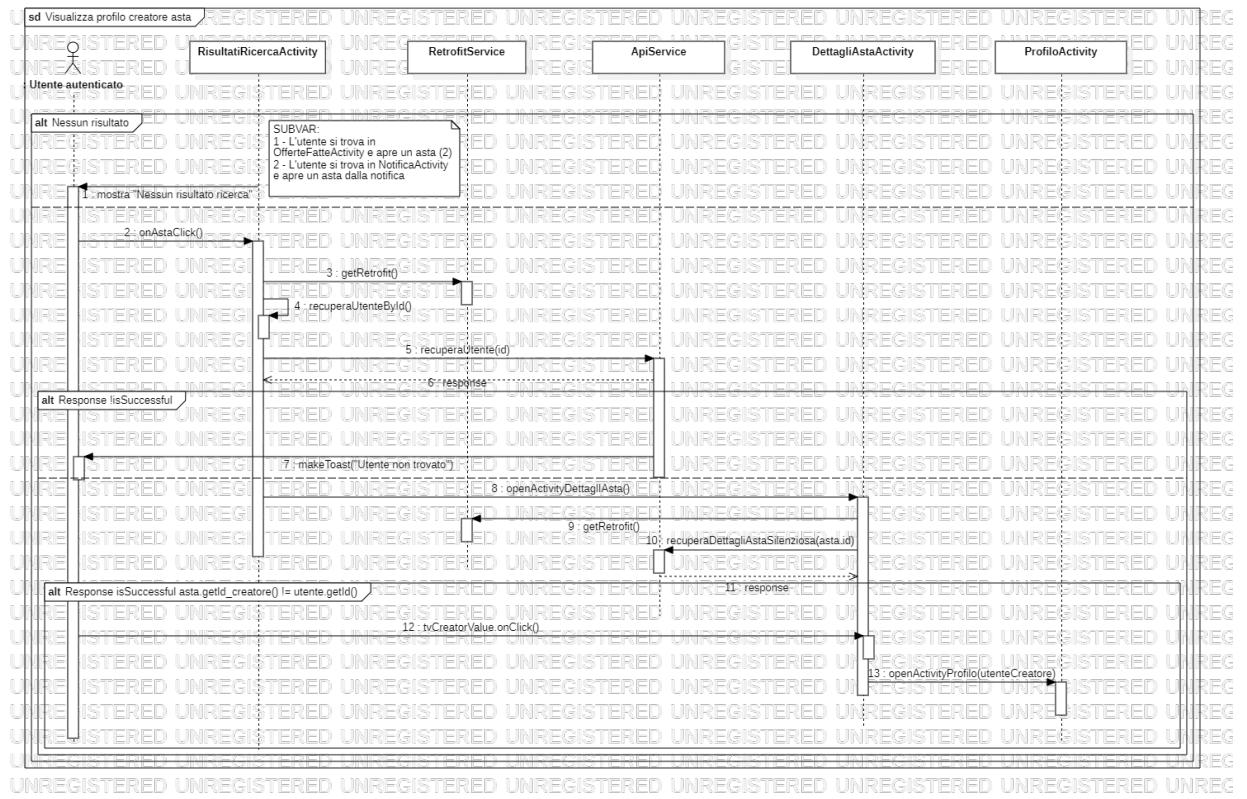
3.3.2 Presentazione Offerta Asta Silenziosa



3.3.3 Ricerca Asta



3.3.4 Visualizzazione Profilo Creatore Asta



4 Codice sorgente

4.1 File di Build automatica

I file di build automatica sono quelli che ci permettono di creare in modo automatico l'infrastruttura software. Nel nostro caso, consideriamo i file:

1. **Dockerfile** del backend
2. **docker-compose.yml**
3. **setup.bat**

```
FROM maven:3.8.4-openjdk-17-slim AS builder
COPY . /app
WORKDIR /app
RUN mvn clean package
FROM openjdk:17-jdk-alpine
COPY --from=builder app/target/dt24-0.0.1-SNAPSHOT.jar app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

Dockerfile Backend

```
version: '3.8'

services:
  postgres:
    image: postgres:14-alpine
    environment:
      POSTGRES_PASSWORD: "dietet_deals"
    ports:
      - "5432:5432"

  backend:
    build: ./backend
    environment:
      SPRING_DATASOURCE_URL: jdbc:postgresql://postgres:5432/dietideals24
    ports:
      - "8080:8080"
```

Docker compose

L'infrastruttura docker che viene a crearsi mediante questo docker-compose avrà due container contenenti l'immagine ufficiale di postgres e l'immagine del backend creata con il dockerfile appena citato. Il client, chiaramente, viene installato a parte.

```
@echo off
REM Setup script for building and running Docker containers

echo =====
echo Building backend Docker image
echo =====

REM Build the backend Docker image
docker build -t backend .
IF %ERRORLEVEL% NEQ 0 (
    echo Error: Failed to build the backend image.
    pause
    exit /b %ERRORLEVEL%
)
echo Backend image built successfully.

echo =====
echo Starting Docker containers
echo =====

REM Run docker-compose to start containers
docker-compose up -d
IF %ERRORLEVEL% NEQ 0 (
    echo Error: Failed to start Docker containers.
    pause
    exit /b %ERRORLEVEL%
)
echo Docker containers started successfully.

echo =====
echo Setup completed successfully
echo =====

REM Wait for the user to press a key before closing
pause
```

setup.bat

Il seguente file, posizionato nella directory del backend, è un eseguibile che rappresenta un semplice script che permette di creare l'immagine del backend ed eseguire il compose del docker-compose.yml automaticamente, creando l'infrastruttura server (in locale). Il file, una volta eseguito, eseguirà le istruzioni citate in modo sequenziale, mostrando errori in caso di insuccesso.

4.2 Versioning

Per lo sviluppo della piattaforma DietiDeals24 è stato adottato GitHub come strumento di versionamento. E' possibile accedere al repository del software al seguente link: <https://github.com/LZannini/DietiDeals24>.

4.3 Report di qualità del codice

Per quanto riguarda l'analisi della qualità del codice abbiamo usato Sonarqube, un utile strumento di analisi statica. Mediante questo software abbiamo identificato e sistemato diversi issue, riuscendo ad ottenere una qualità del codice soddisfacente. Di seguito viene mostrato uno screenshot dei report generati da Sonarqube per il Frontend e Backend:



Report Sonarqube del Frontend (sopra) e Backend (sotto)

5 Testing e Valutazione dell’usabilità

5.1 Test JUnit

Per la progettazione dei casi di test sono stati scelti due metodi nel client e due metodi nel backend. E’ stata adottata la strategia Black Box, perché pur conoscendo la struttura del codice, ci interessava che fossero rispettati i requisiti dell’unità da testare. Per la definizione delle classi di equivalenza è stato utilizzato un approccio R-WECT.

Lato client abbiamo individuato i seguenti metodi da testare: **String calcolaTempoRimanente(LocalDateTime endDateTime, LocalDateTime now)** che serve per il calcolo della scadenza di un’asta per rappresentarla all’interno dell’activity dettagli asta, che ci permette di visualizzare la scadenza di un’asta in un modo più chiaro e leggibile all’interno della scheda riguardo ai dettagli di un’asta. L’altro metodo è **boolean passwordValida(String nuovaPassword, String confermaPassword, String vecchiaPassword)** che serve appunto per verificare che la nuova password sia valida nel momento in cui l’utente decida di cambiarla e ci si assicura che tutte le condizioni siano rispettate.

Nel backend invece, abbiamo individuato i seguenti metodi, il primo è **List<AstaDTO> trovaAstePerParolaChiaveAndCategoria(String chiave, Categoria categoria)** che praticamente si occupa di prendere parola chiave e categoria che l’utente ha inserito nel client e sono arrivate al backend tramite chiamate api e dopo un’interazione con il database tramite AstaRepository, restituisce una lista di aste che corrispondono alla ricerca se ci sono, altrimenti restituisce una lista vuota. L’altro metodo è **boolean verificaOfferta(Utente utente, Asta asta, float valore)** e serve a verificare che l’offerta che sta per essere inserita nel db sia formata correttamente.

5.1.1 calcolaTempoRimanente

Identificazione delle classi di equivalenza del metodo String calcolaTempoRimanente(LocalDateTime endDateTime, LocalDateTime now):

Parametro in Input	Classe di equivalenza	Validità
LocalDateTime endDateTime	CE1: endDateTime != null CE2: endDateTime == null	CE1: Valida CE2: Non valida
LocalDateTime now	CE1: now != null CE2: now == null	CE1: Valida CE2: Non valida

Poi oltre a considerare le classi di equivalenza legate ai singoli parametri, abbiamo considerato anche le classi di equivalenza basate sulle relazioni tra di essi.

Relazione tra i parametri endDateTime e now	Validità
CE1: endDateTime > now	Valida
CE2: endDateTime == now	Valida
CE3: endDateTime < now	Valida

```

public class DettagliAstaActivityTest {
    @Test
    public void testEntrambiNulli() {
        String risultato = DettagliAstaActivity.calcolaTempoRimanente(null ,null);
        assertEquals("Data non valida", risultato);
    }

    @Test
    public void testScadenzaNullaAttualeValida() {
        LocalDateTime dataAttuale = LocalDateTime.now();
        String risultato = DettagliAstaActivity.calcolaTempoRimanente(null ,dataAttuale);
        assertEquals("Data non valida", risultato);
    }

    @Test
    public void testScadenzaValidaAttualeNulla() {
        LocalDateTime dataScadenza = LocalDateTime.now();
        String risultato = DettagliAstaActivity.calcolaTempoRimanente(dataScadenza,null);
        assertEquals("Data non valida", risultato);
    }

    @Test
    public void testEntrambiValidiAttualeDopoScadenza() {
        LocalDateTime dataScadenza = LocalDateTime.now().minusDays(1);
        LocalDateTime dataAttuale = LocalDateTime.now();
        String risultato = DettagliAstaActivity.calcolaTempoRimanente(dataScadenza,dataAttuale);
        assertEquals("Scaduta", risultato);
    }

    @Test
    public void testEntrambiValidiScadenzaMaggioreUnAnno() {
        LocalDateTime dataScadenza = LocalDateTime.now();
        LocalDateTime dataAttuale = LocalDateTime.now().minusYears(2);
        Period period = Period.between(dataAttuale.toLocalDate(), dataScadenza.toLocalDate());
        String risultato = DettagliAstaActivity.calcolaTempoRimanente(dataScadenza,dataAttuale);
        assertEquals(period.getYears() + " anni, " + period.getMonths(), risultato);
    }

    @Test
    public void testEntrambiValidiScadenzaMaggioreUnMese() {
        LocalDateTime dataScadenza = LocalDateTime.now();
        LocalDateTime dataAttuale = LocalDateTime.now().minusMonths(2);
        Period period = Period.between(dataAttuale.toLocalDate(), dataScadenza.toLocalDate());
        String risultato = DettagliAstaActivity.calcolaTempoRimanente(dataScadenza,dataAttuale);
        assertEquals(period.getMonths() + " mesi, " + period.getDays(), risultato);
    }

    @Test
    public void testEntrambiValidiScadenzaMaggioreUnGiorno() {
        LocalDateTime dataScadenza = LocalDateTime.now();
        LocalDateTime dataAttuale = LocalDateTime.now().minusDays(2);
        Period period = Period.between(dataAttuale.toLocalDate(), dataScadenza.toLocalDate());
        String risultato = DettagliAstaActivity.calcolaTempoRimanente(dataScadenza,dataAttuale);
        assertEquals(period.getDays() + " giorni", risultato);
    }

    @Test
    public void testEntrambiValidiScadenzaMaggioreUnOra() {
        LocalDateTime dataScadenza = LocalDateTime.now();
        LocalDateTime dataAttuale = LocalDateTime.now().minusHours(2);
        Duration duration = Duration.between(dataAttuale.toLocalTime(), dataScadenza.toLocalTime());
        String risultato = DettagliAstaActivity.calcolaTempoRimanente(dataScadenza,dataAttuale);
        assertEquals(duration.toHours() + " ore", risultato);
    }

    @Test
    public void testEntrambiValidiScadenzaMaggioreUnMinuto() {
        LocalDateTime dataScadenza = LocalDateTime.now();
        LocalDateTime dataAttuale = LocalDateTime.now().minusMinutes(2);
        Duration duration = Duration.between(dataAttuale.toLocalTime(), dataScadenza.toLocalTime());
        String risultato = DettagliAstaActivity.calcolaTempoRimanente(dataScadenza,dataAttuale);
        assertEquals(duration.toMinutes() + " minuti", risultato);
    }

    @Test
    public void testEntrambiValidiScadenzaMaggioreUnSecondo() {
        LocalDateTime dataScadenza = LocalDateTime.now();
        LocalDateTime dataAttuale = LocalDateTime.now().minusSeconds(1);
        String risultato = DettagliAstaActivity.calcolaTempoRimanente(dataScadenza,dataAttuale);
        assertEquals("meno di un minuto", risultato);
    }

    @Test
    public void testEntrambiValidiScadenzaUgualeAttuale() {
        LocalDateTime dataScadenza = LocalDateTime.now();
        LocalDateTime dataAttuale = LocalDateTime.now();
        String risultato = DettagliAstaActivity.calcolaTempoRimanente(dataScadenza,dataAttuale);
        assertEquals("Scaduta", risultato);
    }
}

```

Codice JUnit per calcolaTempoRimanente

5.1.2 passwordValida

Identificazione delle classi di equivalenza per il metodo boolean passwordValida(String nuovaPassword, String confermaPassword, String vecchiaP):

Parametro in Input	Classe di equivalenza	Validità
nuovaPassword	CE1: uguale a vecchiaPassword CE2: diversa da vecchiaPassword CE3: lunghezza < 8 caratteri CE4: lunghezza \geq 8 caratteri CE5: contiene almeno una maiuscola CE6: non contiene nessuna maiuscola CE7: contiene almeno un numero CE8: non contiene nessun numero CE9: contiene almeno un carattere speciale CE10: non contiene nessun carattere speciale	CE1: Non valida CE2: Valida CE3: Non valida CE4: Valida CE5: Valida CE6: Non valida CE7: Valida CE8: Non valida CE9: Valida CE10: Non valida
confermaPassword	CE1: uguale a nuovaPassword CE2: diversa da nuovaPassword	CE1: Valida CE2: Non valida
vecchiaPassword	CE1: uguale a nuovaPassword CE2: diversa da nuovaPassword	CE1: Non valida CE2: Valida

Sono stati testati anche alcuni casi limite come una nuova password che ha esattamente 8 caratteri, una nuova password molto lunga e una nuova password molto complessa.

```

public class ModificaPasswordActivityTest {

    @Test
    public void testPasswordValida() {
        assertTrue(ModificaPasswordActivity.passwordValida("Password1.", "Password1.", "VecchiaPassword1."));
    }

    @Test
    public void testNuovaUgualeAVecchia() {
        assertFalse(ModificaPasswordActivity.passwordValida("Password1.", "Password1.", "Password1."));
        assertEquals("La nuova password deve essere diversa dalla vecchia password.", ModificaPasswordActivity.getMessaggioErrore());
    }

    @Test
    public void testPasswordCorta() {
        assertFalse(ModificaPasswordActivity.passwordValida("Pass1.", "Pass1.", "Password1."));
        assertEquals("La nuova password deve contenere almeno 8 caratteri.", ModificaPasswordActivity.getMessaggioErrore());
    }

    @Test
    public void testNessunaMaiuscola() {
        assertFalse(ModificaPasswordActivity.passwordValida("password1.", "password1.", "VecchiaPassword1."));
        assertEquals("La nuova password deve contenere almeno una lettera maiuscola.", ModificaPasswordActivity.getMessaggioErrore());
    }

    @Test
    public void testNessunNumero() {
        assertFalse(ModificaPasswordActivity.passwordValida("Password.", "Password.", "VecchiaPassword1."));
        assertEquals("La nuova password deve contenere almeno un numero.", ModificaPasswordActivity.getMessaggioErrore());
    }

    @Test
    public void testNessunCarattereSpeciale() {
        assertFalse(ModificaPasswordActivity.passwordValida("Password1", "Password1", "VecchiaPassword1."));
        assertEquals("La nuova password deve contenere almeno un carattere speciale.", ModificaPasswordActivity.getMessaggioErrore());
    }

    @Test
    public void testNuovaDiversaDaConferma() {
        assertFalse(ModificaPasswordActivity.passwordValida("Password1.", "Password2.", "VecchiaPassword1."));
        assertEquals("Le password non corrispondono, riprova.", ModificaPasswordActivity.getMessaggioErrore());
    }

    @Test
    public void testPasswordComplessa() {
        assertTrue(ModificaPasswordActivity.passwordValida("C0mpl3x!Pass", "C0mpl3x!Pass", "VecchiaPassword1."));
    }

    @Test
    public void testPassword8Caretteri() {
        assertTrue(ModificaPasswordActivity.passwordValida("Pass123.", "Pass123.", "VecchiaPassword1."));
    }

    @Test
    public void testPasswordLunga() {
        String nuovaPassword = "A" + "a".repeat(9998) + "1.";
        assertTrue(ModificaPasswordActivity.passwordValida(nuovaPassword, nuovaPassword, "VecchiaPassword1."));
    }
}

```

Codice JUnit per passwordValida

5.1.3 trovaAstePerParolaChiaveAndCategoria

Identificazione delle classi di equivalenza per il metodo List<AstaDTO> trovaAstePerParolaChiaveAndCategoria(String chiave, Categoria categoria):

Parametro in Input	Classe di equivalenza	Validità
chiave	CE1: chiave != null CE2: chiave == null	CE1: Valida CE2: Non valida
categoria	CE1: oggetto valido di tipo Categoria CE2: categoria == null	CE1: Valida CE2: Non valida

In questo caso sono stati utilizzati anche degli oggetti mock, che sono oggetti finti utili alla simulazione di parti di codice esterne al nostro metodo. Abbiamo usato i mock per simulare il comportamento di astaRepository, ovvero l'oggetto che si occupa dell'interazione con il database.

```

public class AstaServiceImplementsTest {
    @Mock
    private AstaRepository astaRepository;

    @InjectMocks
    private AstaServiceImplements service = new AstaServiceImplements();

    @BeforeEach
    private void setUp() {
        MockitoAnnotations.openMocks(this);
    }

    @Test
    void testTrovaAstePerParolaChiaveAndCategoria_EntrambiNulli() {
        List<AstaDTO> lista = service.trovaAstePerParolaChiaveAndCategoria(null, null);
        assertNull(lista);
    }

    @Test
    void testTrovaAstePerParolaChiaveAndCategoria_ParolaChiaveNulla_CategoriaValida() {
        Categoria categoria = Categoria.ABBIGLIAMENTO;
        List<AstaDTO> lista = service.trovaAstePerParolaChiaveAndCategoria(null, categoria);
        assertNull(lista);
    }

    @Test
    void testTrovaAstePerParolaChiaveAndCategoria_ParolaChiaveValida_CategoriaNulla() {
        String chiave = "T-Shirt";
        List<AstaDTO> lista = service.trovaAstePerParolaChiaveAndCategoria(chiave, null);
        assertNull(lista);
    }

    @Test
    void testTrovaAstePerParolaChiaveAndCategoria_ParolaChiaveValida_CategoriaValida_AsteTrovate() {
        String chiave = "T-Shirt";
        Categoria categoria = Categoria.ABBIGLIAMENTO;
        List<Asta> aste = Arrays.asList(new Asta_Ribasso(), new Asta_Silenziosa(), new Asta_Inversa());
        when(astaRepository.filtraPerCategoriaAndParoleChiave(chiave, categoria, StatoAsta.ATTIVA)).thenReturn(aste);

        List<AstaDTO> risultato = service.trovaAstePerParolaChiaveAndCategoria(chiave, categoria);

        assertNotNull(risultato);
        assertEquals(3, risultato.size());
        assertEquals("RIBASSO", risultato.get(0).getTipo());
        assertEquals("SILENZIOSA", risultato.get(1).getTipo());
        assertEquals("INVERSA", risultato.get(2).getTipo());

        verify(astaRepository, times(1))
            .filtraPerCategoriaAndParoleChiave(chiave, categoria, StatoAsta.ATTIVA);
    }

    @Test
    void testTrovaAstePerParolaChiaveAndCategoria_ParolaChiaveValida_CategoriaValida_AsteNonTrovate() {
        String chiave = "T-Shirt";
        Categoria categoria = Categoria.ABBIGLIAMENTO;

        when(astaRepository.filtraPerCategoriaAndParoleChiave(chiave, categoria, StatoAsta.ATTIVA)).thenReturn(Collections.emptyList());

        List<AstaDTO> risultato = service.trovaAstePerParolaChiaveAndCategoria(chiave, categoria);

        assertNotNull(risultato);
        assertTrue(risultato.isEmpty());
    }
}

```

Codice JUnit per trovaAstePerParolaChiaveAndCategoria

5.1.4 verificaOfferta

Identificazione delle classi di equivalenza per il metodo boolean verificaOfferta(Utente utente, Asta asta, float valore):

Parametro in Input	Classi di equivalenza	Validità
utente	CE1: utente != null CE2: utente == null	CE1: Valida CE2: Non valida
asta	CE1: asta == null CE2: asta.getStato() == ATTIVA CE3: asta.getStato() != ATTIVA	CE1: Non valida CE2: Valida CE3: Non valida
valore	CE1: valore >= 0 CE2: valore < 0	CE1: Valida CE2: Non valida

```

public class OffertaServiceImplementsTest {
    @InjectMocks
    private OffertaServiceImplements service = new OffertaServiceImplements();

    private Utente utenteCompratore;
    private Utente utenteVenditore;
    private Asta_Inversa astaInversa;
    private Asta_Ribasso astaRibasso;
    private Asta_Silenziosa astaSilenziosa;
    private Asta astaNonAttiva;

    @BeforeEach
    void setUp() {
        utenteCompratore = new Utente("Utente1", "Email1", "Pass1", TipoUtente.COMPRATORE);
        utenteVenditore = new Utente("Utente2", "Email2", "Pass2", TipoUtente.VENDITORE);

        astaInversa = new Asta_Inversa(StatoAsta.ATTIVA);
        astaRibasso = new Asta_Ribasso(StatoAsta.ATTIVA);
        astaSilenziosa = new Asta_Silenziosa(StatoAsta.ATTIVA);
        astaNonAttiva = new Asta(StatoAsta.FALLITA);
    }

    @Test
    void testVerificaOfferta_OffertaValida_AstaInversa() {
        assertTrue(service.verificaOfferta(utenteVenditore, astaInversa, 50));
    }

    @Test
    void testVerificaOfferta_OffertaValida_AstaRibasso() {
        assertTrue(service.verificaOfferta(utenteCompratore, astaRibasso, 50));
    }

    @Test
    void testVerificaOfferta_OffertaValida_AstaSilenziosa() {
        assertTrue(service.verificaOfferta(utenteCompratore, astaSilenziosa, 50));
    }

    @Test
    void testVerificaOfferta_AstaNulla() {
        IllegalStateException exception = assertThrows(IllegalStateException.class, () -> {
            service.verificaOfferta(utenteCompratore, null, 50);
        });
        assertEquals("L'asta non è più attiva.", exception.getMessage());
    }

    @Test
    void testVerificaOfferta_UtenteNullo() {
        IllegalStateException exception = assertThrows(IllegalStateException.class, () -> {
            service.verificaOfferta(null, astaRibasso, 50);
        });
        assertEquals("L'utente non è stato trovato.", exception.getMessage());
    }

    @Test
    void testVerificaOfferta_AstaNonAttiva() {
        IllegalStateException exception = assertThrows(IllegalStateException.class, () -> {
            service.verificaOfferta(utenteCompratore, astaNonAttiva, 50);
        });
        assertEquals("L'asta non è più attiva.", exception.getMessage());
    }

    @Test
    void testVerificaOfferta_ValoreZero() {
        IllegalStateException exception = assertThrows(IllegalArgumentException.class, () -> {
            service.verificaOfferta(utenteCompratore, astaSilenziosa, 0);
        });
        assertEquals("Il valore dell'offerta è inferiore al minimo consentito.", exception.getMessage());
    }

    @Test
    void testVerificaOfferta_ValoreMinoreDiZero() {
        IllegalStateException exception = assertThrows(IllegalArgumentException.class, () -> {
            service.verificaOfferta(utenteCompratore, astaSilenziosa, -1);
        });
        assertEquals("Il valore dell'offerta è inferiore al minimo consentito.", exception.getMessage());
    }

    @Test
    void testVerificaOfferta_InversaCompratore() {
        IllegalStateException exception = assertThrows(IllegalStateException.class, () -> {
            service.verificaOfferta(utenteCompratore, astaInversa, 50);
        });
        assertEquals("In un'asta inversa, il compratore non può partecipare.", exception.getMessage());
    }

    @Test
    void testVerificaOfferta_RibassoVenditore() {
        IllegalStateException exception = assertThrows(IllegalStateException.class, () -> {
            service.verificaOfferta(utenteVenditore, astaRibasso, 50);
        });
        assertEquals("In un'asta a ribasso o silenziosa, il venditore non può partecipare come offerente.", exception.getMessage());
    }

    @Test
    void testVerificaOfferta_SilenziosaVenditore() {
        IllegalStateException exception = assertThrows(IllegalStateException.class, () -> {
            service.verificaOfferta(utenteVenditore, astaSilenziosa, 50);
        });
        assertEquals("In un'asta a ribasso o silenziosa, il venditore non può partecipare come offerente.", exception.getMessage());
    }
}

```

Codice JUnit per verificaOfferta

5.2 Valutazione dell'usabilità sul campo

5.2.1 Test di usabilità

Questi tipi di test sono condotti con il sistema completamente funzionante. E' importante conoscere le tipologie di utenti che vanno a svolgere questi test per sapere che grado di familiarità hanno con la tecnologia e la loro conoscenza del dominio del sistema. Abbiamo considerato 4 tipologie diversi di utenti:

1. **Anna**, bassa familiarità con la tecnologia e bassa conoscenza del dominio del sistema.
2. **Gianni**, alta familiarità con la tecnologia e bassa conoscenza del dominio del sistema.
3. **Leonardo**, bassa familiarità con la tecnologia e alta conoscenza del dominio del sistema.
4. **Francesca**, alta familiarità con la tecnologia e alta conoscenza del dominio del sistema.

Sono rappresentati nella seguente tabella gli utenti e i compiti che devono svolgere, per misurare questi test useremo 3 categorie come fatto per l'usabilità a priori, Successo (S), Fallimento (F), Successo Parziale (P).

	Sign-In	Modifica informazioni del profilo	Creazione asta silenziosa	Ricerca per categoria	Visualizza offerte per una determinata asta creata	Presenta offerta per un'asta al ribasso
Anna	S	P	S	S	F	P
Gianni	S	S	S	S	S	P
Leonardo	S	S	S	S	P	S
Francesca	S	S	S	S	S	S

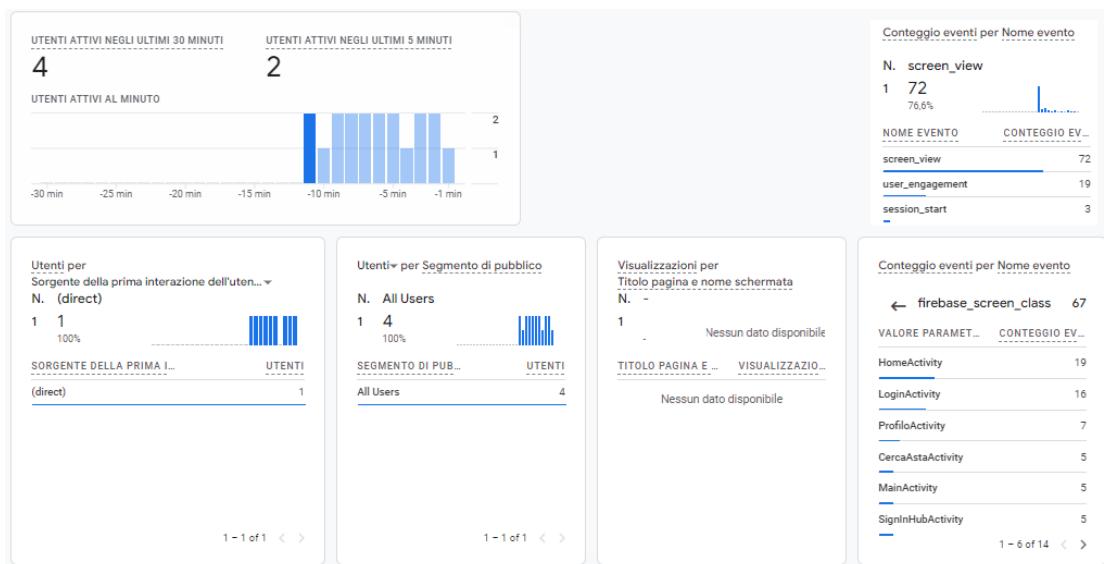
$$\text{Tasso di successo} = \frac{18 + (5 \times 0,5)}{24} = 88\%$$

Dalla tabella vediamo che su 24 compiti, 19 sono stati eseguiti con successo (S), 4 sono stati eseguiti solo in parte (P) e in 1 caso invece c'è stato un fallimento (F). In questa formula ogni successo parziale è stato conteggiato come la metà di un successo pieno.

5.2.2 Report Google Analytics

Per la valutazione dell'usabilità sul campo, oltre al test di usabilità abbiamo deciso di effettuare un analisi con Google Analytics. In particolare, abbiamo usato **Google Analytics for Firebase**, uno strumento di analisi offerto da Google, integrato nella piattaforma Firebase, che consente agli sviluppatori di applicazioni mobile di monitorare e analizzare il comportamento degli utenti all'interno delle loro applicazioni.

Nello specifico, abbiamo riportato una panoramica degli utenti che hanno usato l'applicativo negli ultimi 30 minuti (dal report), specificando il tipo di evento e le Activity più visualizzate.



Report Google Analytics

*Loris Zannini
Mattia Marucci
Vincenzo Meloni*