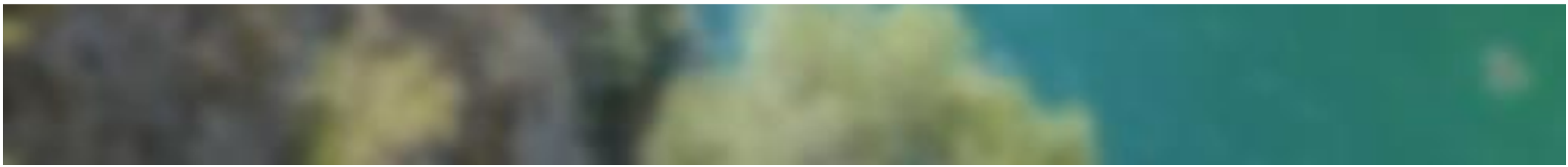
DIETIESTATES25

UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II
A.A. 2024/25

Team INGSW2425_055

VISCONTI GAETANO
N86002282

BORGIA MANUELE
N86003557



Indice

(Pagina lasciata intenzionalmente vuota)

1 Introduzione al documento

DietiEstates25 è un prodotto software con l'obiettivo di fornire, al target di utenti che lo utilizza, un'esperienza soddisfacente nella ricerca di immobili in vendita o affitto consentendo loro di confrontare e valutare le offerte proposte da diverse agenzie, mettersi in contatto e, in caso di particolare interesse, contrattare con chi ha in gestione il bene.

1.1 Panoramica sui prossimi capitoli

Questa sezione ha il fine di sintetizzare la struttura complessiva del documento illustrandone i capitoli che spiegheranno il processo per la realizzazione di un prodotto software, per ordine:

- **Documento dei requisiti software:** Questo capitolo serve a spiegare in modo dettagliato il processo iterativo che consente di elicitare, specificare e validare i requisiti che serviranno a distinguere i possibili utilizzatori di DietiEstates25.
  In una successiva fase, con alcuni strumenti visivi, sarà già possibile vedere il risultato atteso.

- **Documento di design del sistema:** Questo capitolo concentra la sua attenzione sulle scelte progettuali che incideranno sulla successiva fase di sviluppo del software. Nel dettaglio (ed in più sezioni) verranno giustificate da un'accurata documentazione UML l'architettura adottata, gli altri strumenti impiegati, le convenzioni adottate.

- **Documento sul processo di sviluppo:** Questo capitolo illustra il file di build automatica, file che consente l'uso del progetto a prodotto finito. Altre sezioni spiegheranno gli strumenti di versioning impiegati e il report sulla qualità del codice.

- **Documento sul testing e sulla valutazione dell'usabilità:** Questo capitolo, sulla base di requisiti funzionali sviluppati nel capitolo precedente, individua quattro funzioni e per ognuna di essi si selezionano i suoi parametri, si partiziona il dominio di input e si applica un criterio di test (si applicherà la copertura minima considerabile), infine si scelgono dei valori per il blocco di test.
  Un'altra sezione del documento si concentra sull'usabilità e la curva di apprendimento che un utente (generico o specifico) ha nel momento in cui usa il prodotto software.

2 Documento dei Requisiti Software

In questo capitolo verranno raccolti tutti i requisiti (utente e di sistema) necessari alla definizione di quello che dovrà essere, a lavoro concluso, il prodotto software DietiEstates25.

2.1 Glossario

In questa sezione sono elencati i termini utilizzati nel documento che segue e le relative definizioni:

- **Admin (anche chiamato Amministratore):** Questa categoria di utente ha la possibilità di amministrare il prodotto DietiEstates25 e di aggiungere nuovi amministratori di supporto (vedi dopo) e nuovi agenti immobiliari (vedi dopo).
- **Amministratore di supporto:** Come l'admin, l'amministratore di supporto ha la possibilità di gestire DietiEstates25 e di aggiungere (solo) nuovi agenti immobiliari (vedi definizione successiva).
- **Agente immobiliare:** Un agente immobiliare è l'utente (assieme all'admin e gli amministratori di supporto) che può aggiungere immobili, gestire le offerte per essi ed interloquire con i clienti (a seguire la definizione).
- **API (Application Programming Interface):** Insieme di regole, protocolli e strumenti che permette a due componenti software di comunicare tra loro. Definisce il modo in cui un'applicazione può richiedere servizi o dati a un'altra. Nel contesto di DietiEstates25, le API vengono utilizzate per integrare servizi esterni (es. mappe, autenticazione) e per far interagire il front-end con il back-end.
- **API Places:** Servizio API specifico (offerto da Google) che fornisce l'accesso ad un database di luoghi e punti di interesse (POI). Utilizzato in DietiEstates25 per consentire di visualizzare o servizi (es: scuole, parchi, fermate dell'autobus) vicini ad un immobile, migliorando l'esperienza di ricerca per il cliente.
- **AWS:** Amazon Web Services. Piattaforma di cloud computing on-demand utilizzata per ospitare, distribuire e gestire applicazioni e servizi, come l'infrastruttura back-end di DietiEstates25.
- **Cliente:** Il cliente è la categoria di utente che utilizza DietiEstates25 per la ricerca di un immobile e che ha possibilità di contrattare col detentore del bene proponendo una offerta iniziale (vedi dopo).
- **Cognito:** Servizio di AWS che fornisce funzionalità di autenticazione e gestione degli utenti. Utilizzato nell'applicazione per gestire il login e la registrazione sicura di amministratori, agenti e clienti.

- **Dashboard:** Interfaccia utente riservata agli utenti autenticati. Essa presenta due sezioni, una (sopra) per la ricerca, l'altra (sotto) per la visualizzazione dei risultati.
- **DietiEstates25:** Nome del progetto e del prodotto software. Si riferisce alla piattaforma di intermediazione immobiliare che facilita le trattative di compravendita tra agenti e clienti.
- **Immobile:** L'elemento centrale di DietiEstates25. Esso è il risultato ottenuto dalla ricerca e dalle preferenze impostate dall'utente mostrandone il prezzo sul mercato, dettagli testuali e immagini. Da qui partono ed arrivano tutte le iterazioni fra agenti e clienti.
- **Mock-up:** Rappresentazione statica e visiva dell'interfaccia utente del prodotto. Mostra la disposizione orientativa degli elementi grafici, i colori e la tipografia, senza funzionalità interattive.
- **Offerta iniziale:** Essa indica l'offerta che un cliente ha interesse nel proporre nei confronti di un agente per provare a contrattare sul bene di interesse.
- **PgAdmin:** Strumento di amministrazione e sviluppo per il database PostgreSQL. Utilizzato per gestire e modificare il database dell'applicazione.
- **Prototipo:** Modello interattivo e funzionale che serve a presentare orientativamente un'applicazione. Esso simula il flusso utente permettendo di testare le funzionalità prima dello sviluppo finale.
- **Risposta offerta:** Essa indica la risposta che un agente dà ad un cliente a seguito di una offerta iniziale.
- **SonarQube:** Piattaforma open-source per il controllo continuo della qualità del codice. Analizza automaticamente il codice sorgente per rilevare bug, vulnerabilità di sicurezza, "code smells" e per misurare la copertura dei test, aiutando gli sviluppatori a mantenere un codice pulito e affidabile.
- **RD:** Ossia Requisito di Dominio. Esso, in questo contesto, indica le regole specifiche del dominio immobiliare.
- **RNF:** Ossia Requisito Non Funzionale. Essi servono a definire i vincoli sul sistema e la sua qualità.
- **Stakeholders:** Portatore di interesse. Nel contesto del progetto, gruppo di persone che ha un interesse (commerciale, tecnico, d'uso) nel successo di DietiEstates25 aiutando, mediante delle interviste (formali o informali), a definire i requisiti del sistema da sviluppare.
- **UCD:** Acronimo di Use Case Diagram ovvero Diagramma dei Casi d'Uso. In questo contesto è un approccio semi-formale per la specifica di requisiti funzionali.
- **UML:** Unified Modeling Language. Linguaggio di modellazione grafico standardizzato utilizzato in questo progetto per modellare la struttura (classi) e il comportamento (casi d'uso, sequenze) del sistema DietiEstates25.

- **xUnit:** Famiglia di framework di testing per linguaggi di programmazione, basata sul pattern originario di JUnit. Nel contesto di DietiEstates25, si riferisce specificatamente al framework per il linguaggio C# (.NET) utilizzato per scrivere ed eseguire test unitari automatizzati sul codice del progetto.

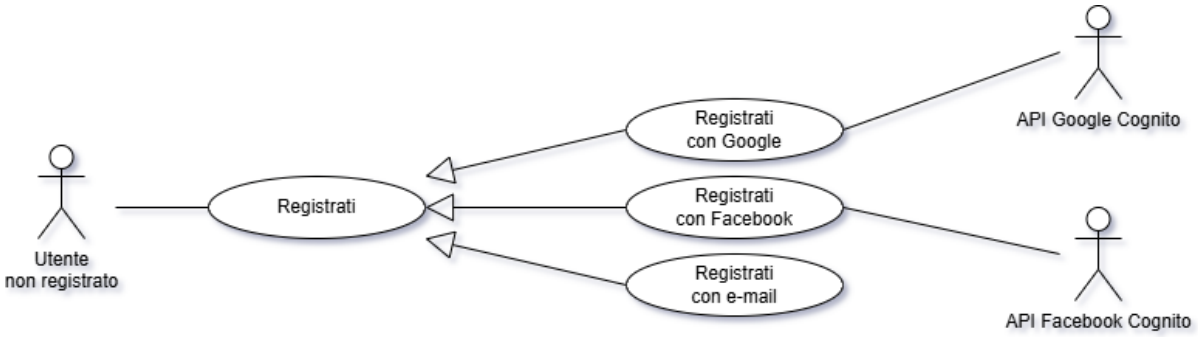
2.2 Modello dei Casi d'Uso

In questa sezione iniziamo ad entrare nel pieno della documentazione iniziando ad illustrare, mediante degli strumenti visivi, quelli che sono i requisiti funzionali.

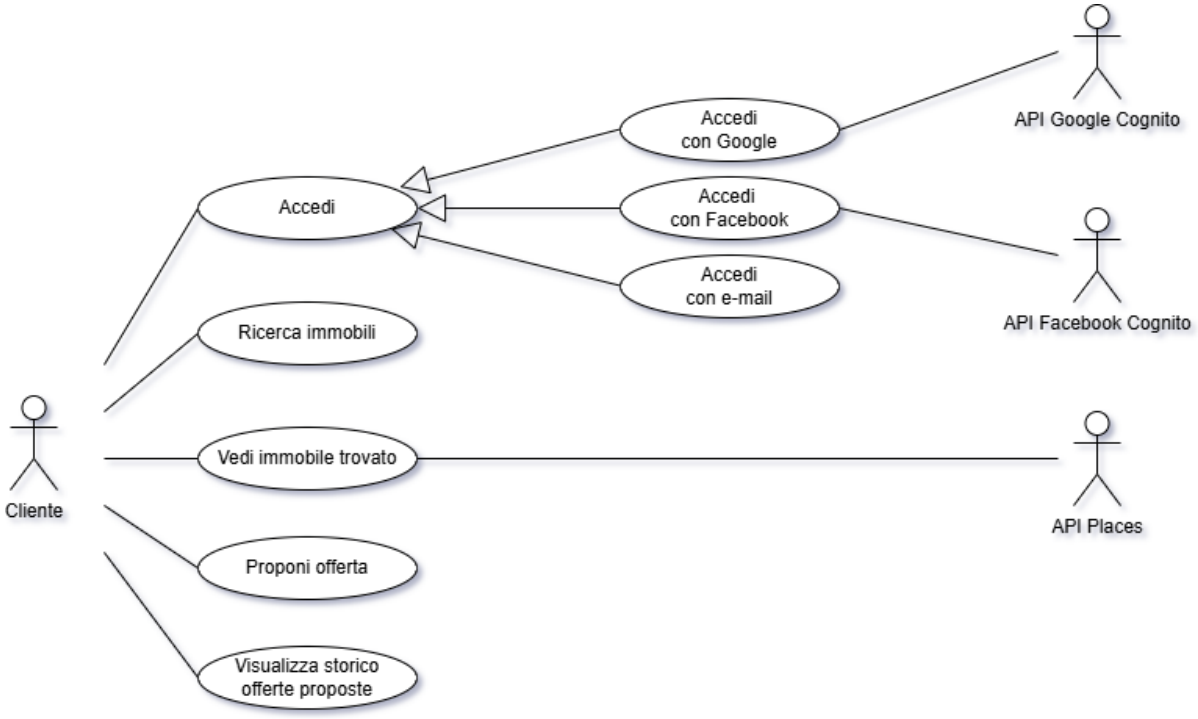
I casi d'uso sono un modo per descrivere le interazioni fra utenti ed il sistema utilizzando un modello grafico ed un testo strutturato in linguaggio naturale.

Il tool che è stato utilizzato per la realizzazione degli Use Case Diagram (abbreviato con UCD) si chiama "draw.io".

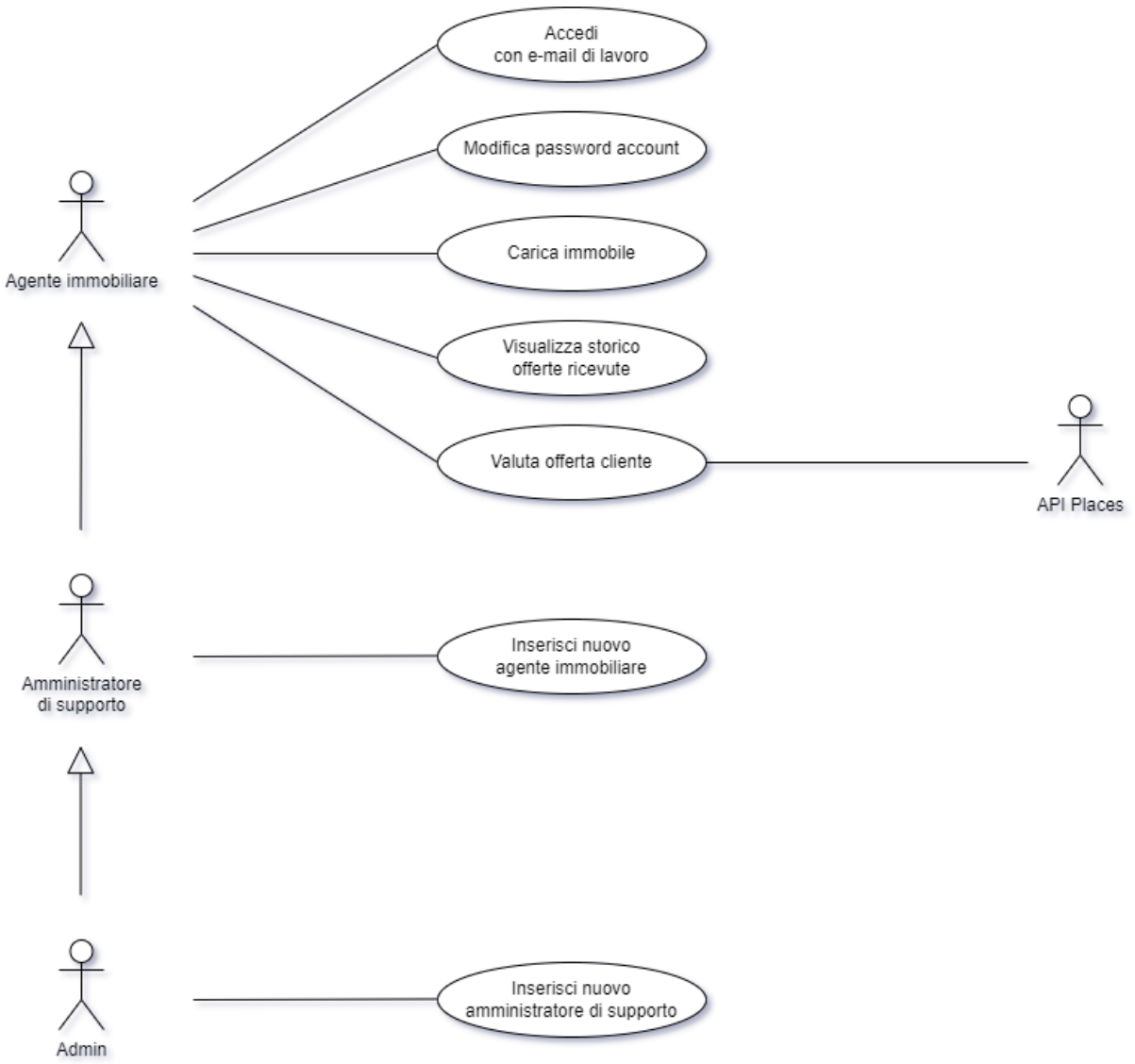
UCD di un utente non registrato:



Il seguente UCD mostra cosa può fare un utente se ha già effettuato la registrazione a DietiEstates25:



Quest’ultimo UCD invece mostra cosa può fare il sistema se ad accedere alla piattaforma sono agenti, amministratori di supporto o admin:



2.3 Individuazione del target utenti

Prima di progettare un prodotto software è indispensabile conoscere gli utenti che utilizzeranno l'applicazione e l'analisi dei requisiti utente è fondamentale per permettere al sistema di soddisfare i diversi obiettivi richiesti.

La progettazione delle cosiddette “Personas” ha come obiettivo quello di conoscere gli utenti finali che utilizzeranno il sistema informativo, questi aiutano a scoprire quelli che saranno i requisiti utente.

Lo strumento impiegato per la realizzazione di essi è “Figma”.

Il sistema in questione consta di due macrocategorie che sono così riassumibili:

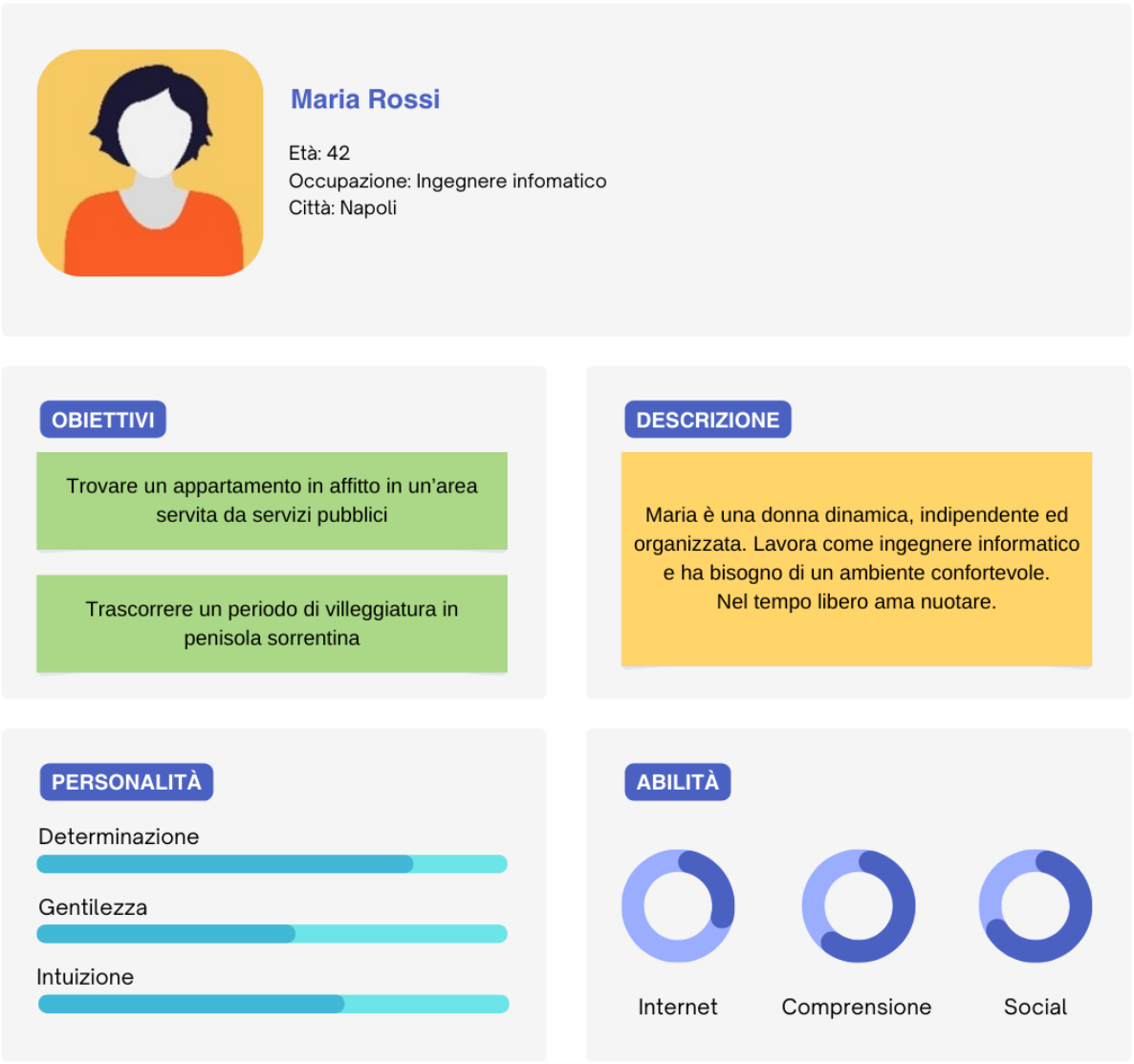
- **Clienti:** Questa categoria di utenti è rappresentata da qualunque persona che, a seconda dei propri interessi, voglia usufruire dei servizi del sistema.
  Questi avranno la possibilità di visualizzare tutti gli immobili presenti sul territorio ed eventualmente acquistare o prendere in affitto.
  Ogni cliente, in caso di interesse per un immobile, ha la possibilità di proporre un'offerta.
  Per ogni cliente, inoltre, sarà garantita una rapida curva di apprendimento del software così che ognuno possa navigare agevolmente e senza grosse difficoltà.

- **Agenti Immobiliari:** Questa categoria di utenti è rappresentata da tutte le persone che utilizzeranno il sistema per scopi lavorativi.
  Ogni agente immobiliare che si troverà ad usufruire del software potrà creare inserzioni immobiliari e valutare le offerte proposte dai clienti.
  Per ogni agente immobiliare, inoltre, sarà garantita un'interazione professionale e chiara col pubblico.

Di seguito sono riportati quattro archetipi ipotetici degli utenti effettivi.

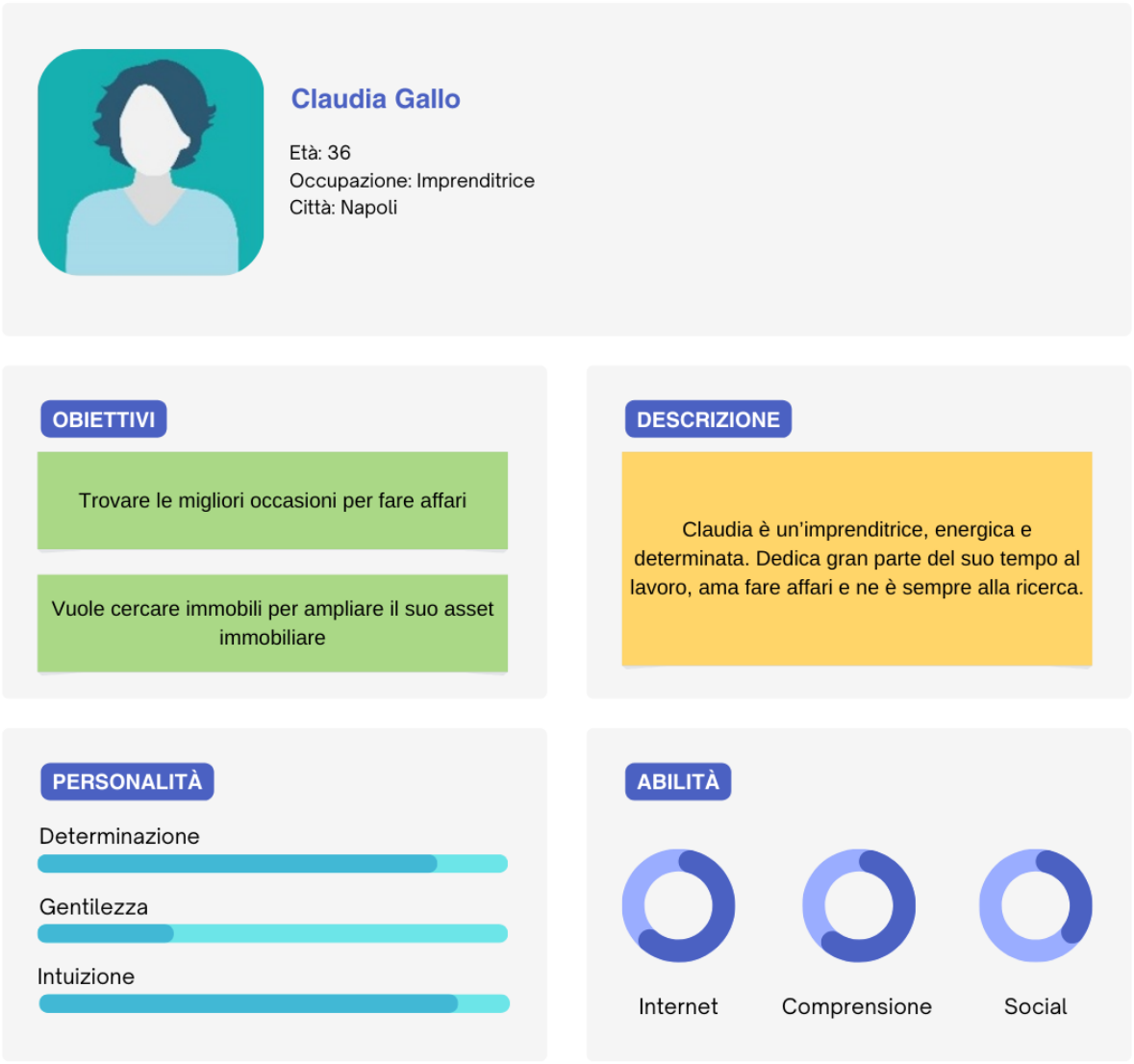
2.3.1 Personas di Maria Rossi

Maria Rossi, utente molto esperta, utilizzerà il sistema per un motivo ben specifico, necessita di un appartamento in affitto per una villeggiatura concedendosi un po' di relax.
Il sistema dovrà offrire a Maria la possibilità farle fare una scelta ottimale dettagliata dalle sue preferenze.
Le poche interazioni con il sistema dovrebbero essere sufficienti per farle trovare la proposta migliore.



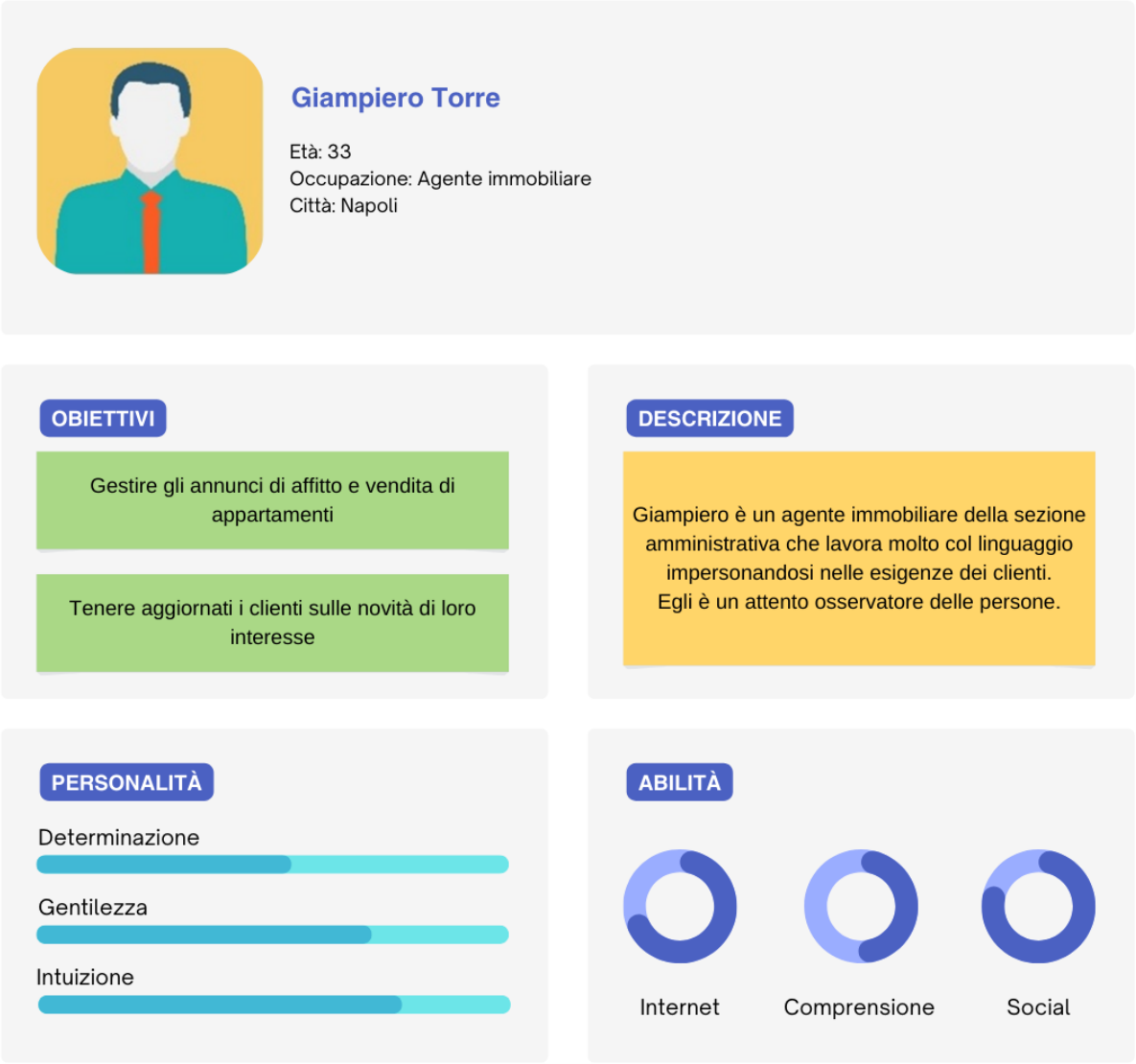
2.3.2 Personas di Claudia Gallo

Imprenditrice affermata, utilizzerà il sistema principalmente per affari.
Claudia, tra i suoi obiettivi, vuole ampliare il suo asset immobiliare.
Il sistema permetterà a Claudia di ricevere le informazioni più dettagliate possibili per i suoi scopi e di permetterle di lavorare con comodità di gestione ed efficienza.



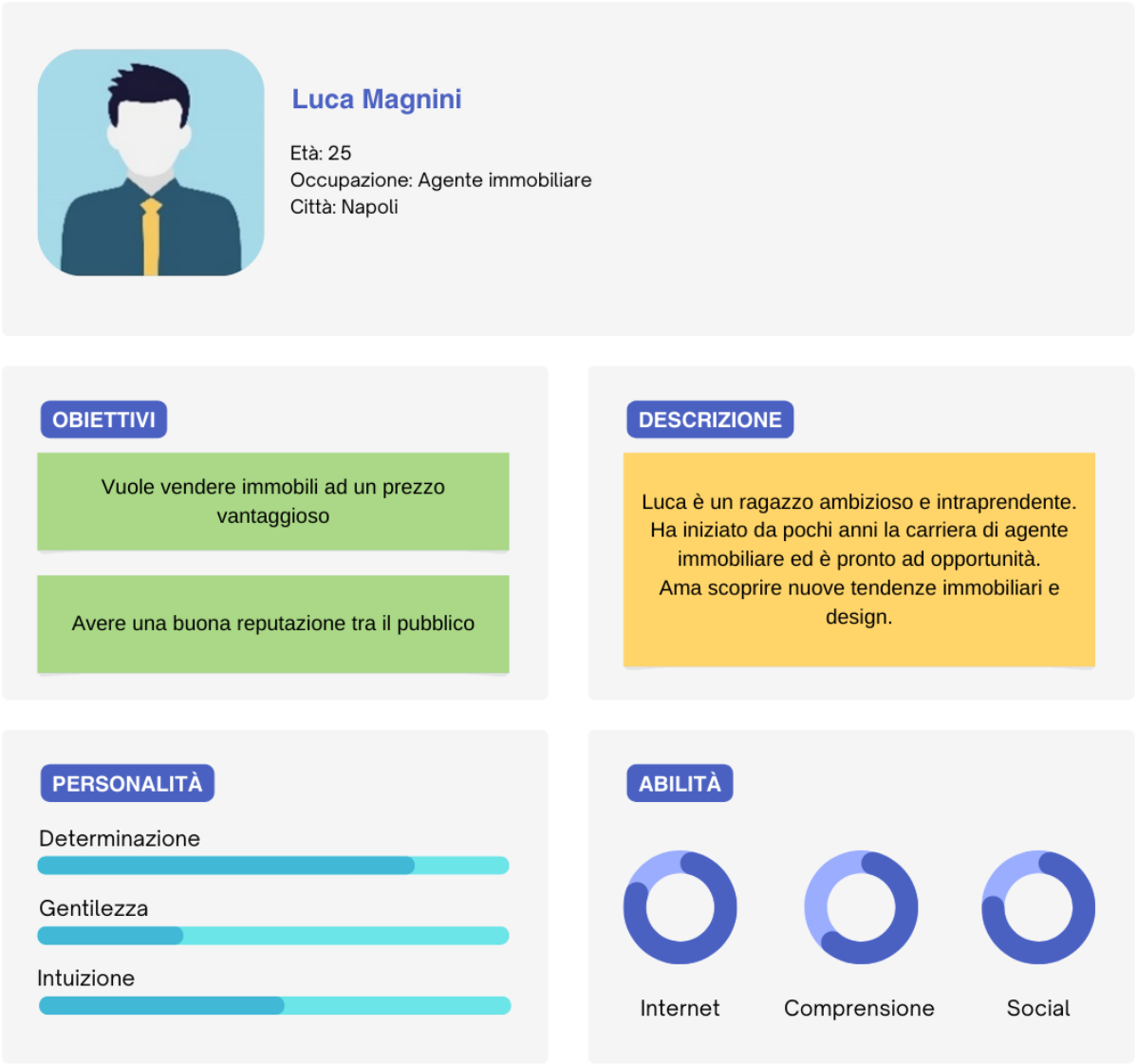
2.3.3 Personas di Giampiero Torre

Agente Immobiliare di esperienza, interagirà col sistema per scopi lavorativi.
Giampiero lavora molto col pubblico ed è interessato alle esigenze dei clienti che interagiscono col software.
Non lavorando spesso attraverso un software, Giampiero avrà una curva di apprendimento più alta rispetto agli altre ragioni per cui il sistema dovrà facilitargli il più possibile l'interazione.



2.3.4 Personas di Luca Magnini

Giovane agente immobiliare con pochi anni di carriera, utilizzerà il software principalmente per vendere o affittare gli immobili. Il sistema garantirà un rapido apprendimento delle sue funzionalità. Le sue abilità informatiche ne faciliteranno l'apprendimento.



2.4 Requisiti non-funzionali e di dominio

In questa sezione verrà presentato un elenco di requisiti non funzionali (abbreviati con la sigla RNF) e un successivo elenco di requisiti di dominio (abbreviati con la sigla RD).

2.4.1 Requisiti non-funzionali

L'elenco che segue definisce tutti quei requisiti che descrivono i vincoli del sistema.

2.4.1.1 Requisiti del prodotto

RNF-PROD-01: (Usabilità) Un cliente principiante, dopo aver esplorato autonomamente il prodotto per circa 10 minuti, deve essere in grado di:

- ricercare immobili
- proporre offerte per un immobile
- visualizzare lo storico delle offerte proposte

RNF-PROD-02: (Usabilità) Un agente immobiliare principiante, dopo una sessione esplorativa di 30 min, deve essere in grado di:

- ricercare immobili
- aggiungere un nuovo immobile
- visualizzare e gestire lo storico delle offerte ricevute

RNF-PROD-03: (Affidabilità) In caso di inserimento di dati non validi il sistema continua a funzionare.

RNF-PROD-04: (Performance) Le operazioni di navigazione principale nell'applicativo (come la ricerca di un immobile) devono completarsi in un tempo percepito dall'utente come immediato.

2.4.1.2 Requisiti organizzativi

RNF-ORG-01: (Processo) Il sistema deve essere sviluppato utilizzando un linguaggio di programmazione Object-Oriented.

RNF-ORG-02: (Qualità) La qualità del codice del back-end deve essere analizzata con SonarQube, Il codice deve essere mantenuto privo di bugs e vulnerabilità.

RNF-ORG-03: (Deployment) L'architettura deve essere progettata per la distribuzione su servizi cloud pubblici.

RNF-ORG-04: (Deployment) Il back-end deve essere indipendente dal front-end.

RNF-ORG-05: (Sviluppo) Il codice sorgente deve essere versionato utilizzando Git con commit significativi.

RNF-ORG-06: (Manutenibilità) Il codice deve seguire le convenzioni di naming di Java e includere commenti Javadoc per tutte le classi pubbliche.

2.4.1.3 Requisiti esterni

RNF-EXT-01: (Sicurezza) Le password degli utenti devono essere memorizzate in modo sicuro.

2.4.2 Requisiti di dominio

L'elenco che segue definisce, invece, i requisiti che appartengono al settore immobiliare.

RD-01: Un immobile, una volta creato, è categorizzato alla vendita o all'affitto.

RD-02: Un'offerta iniziale proposta da un cliente su un immobile può specificare un prezzo solo inferiore al prezzo dell'annuncio. Il sistema non accetta offerte superiori o uguali a quella proposta dall'agente.

RD-03: La modifica dell'inserzione immobiliare è consentita solo all'agente / amministratore di supporto o admin che l'ha inserita.

RD-06: I prezzi degli immobili devono essere espressi in euro.

2.5 Prototipazioni visuali & descrizioni testuali strutturate

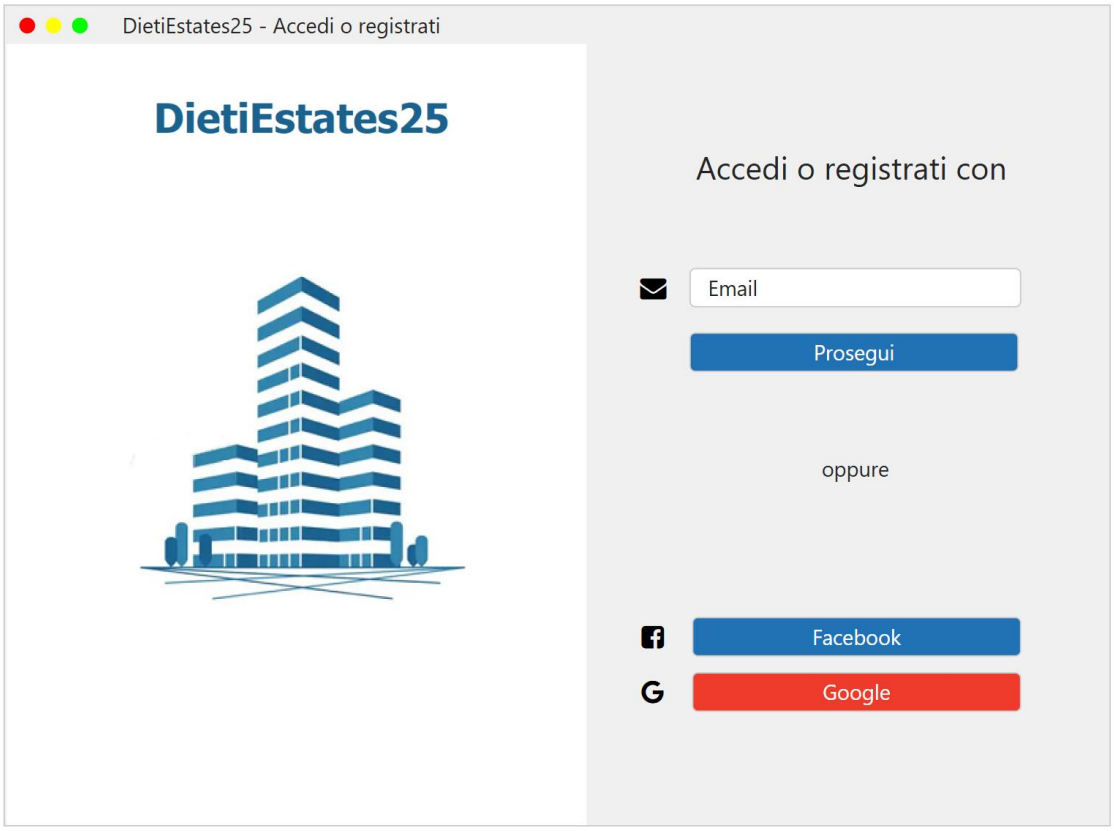
In questa sezione verranno presentati e descritti sei casi d'uso e, per ognuno di essi, sarà prima illustrato con un Mockup seguita da una tabella di Cockburn che ne descrive gli step del caso d'uso in esame.

Nota di precisazione prima della presentazione dei contenuti che seguono:
I Mockup presentati nelle prossime sottosezioni sono a "bassa fedeltà" ossia rappresentano l'idea progettuale e l'aspirazione finale per l'interfaccia utente. Essi visualizzano in modo chiaro il layout, il flusso di navigazione e la disposizione degli elementi.
È importante notare che, per lo sviluppo del prototipo funzionante, è stata utilizzata la tecnologia Java Swing. Questa scelta tecnologica, sebbene solida e cross-platform, presenta vincoli grafici differenti rispetto agli strumenti di prototipazione.
Pertanto, l'implementazione finale, pur rispettando fedelmente la struttura logica e l'intento progettuale qui illustrato, potrà presentare differenze nell'estetica dettagliata.

Il tool utilizzato per la realizzazione dei mockup si chiama "mydraft.cc".

2.5.1 Mockup: Utente non registrato effettua registrazione tramite Google

Mockup 0



A seguito della pressione del bottone “Google” si apre il browser predefinito che mostra la schermata per l’accesso con un account Google.
Una volta completati gli step del login con Google si ritorna su DietiEstate25 per completare la procedura di registrazione mostrando il seguente mockup:

Alla pressione del bottone “Conferma accesso” si ha:

Mockup C1

2.5.1 Cockburn: Utente non registrato effettua registrazione tramite Google

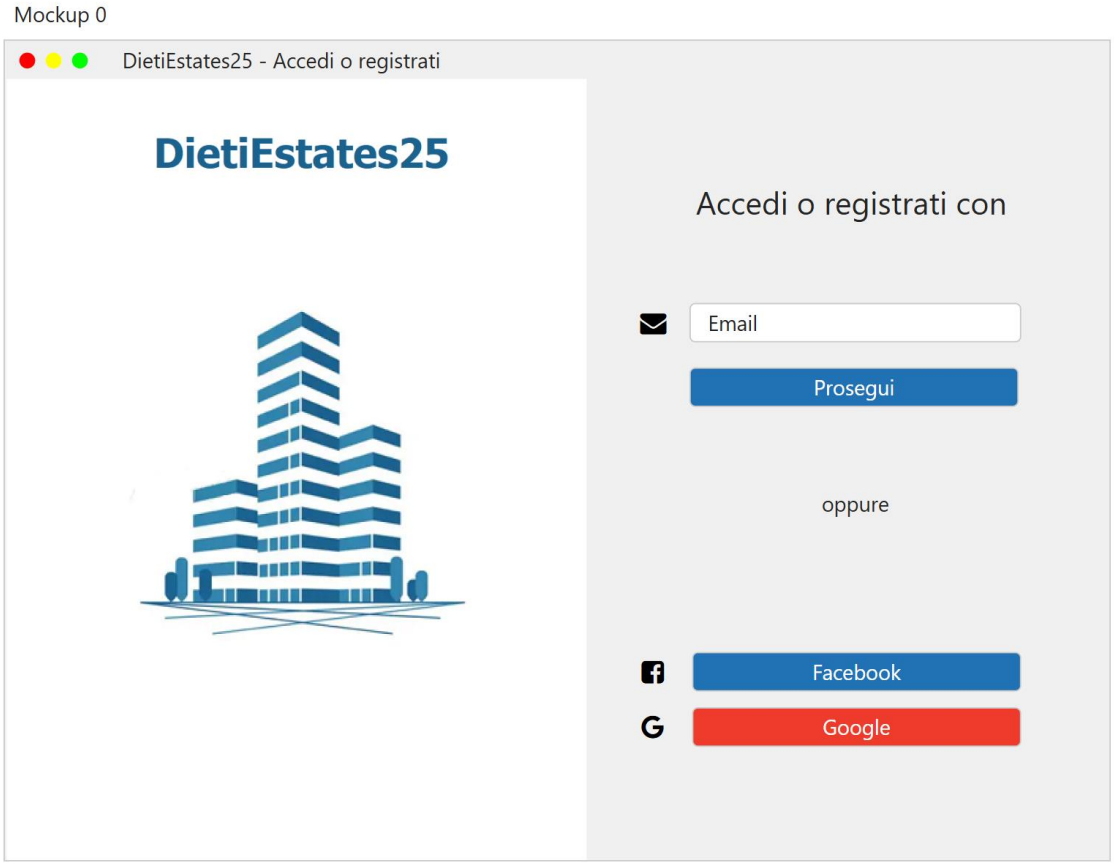
USE CASE #1	Registrati con Google			
Obiettivo	L'Utente vuole registrarsi tramite l'account di Google			
Precondizioni	L'Utente ha aperto il software			
Condizione di successo	L'Utente completerà la registrazione ed avrà accesso alle funzionalità dei Clienti			
Condizione di fallimento	L'Utente non riuscirà a registrarsi, non avrà accesso alle funzionalità dei Clienti			
Attore primario	Utente non registrato			
Trigger	L'Utente preme pulsante "Google'' in M0			
Main Scenario	**Step n.**	**Utente non registrato**	**API Google Cognito**	**Sistema**
	1	Preme il pulsante "Google'' in M0		
	2			Reindirizza l'Utente all'API Places
	3		Mostra schermata di accesso Google	
	4	Sceglie account e inserisce credenziali di Google		
	5		Convalida l'accesso e richiede Autorizzazione di condivisione dati	
	6	Autorizza condivisione di dati Google		
	7		Genera token di accesso	
	8			Riceve token e completa registrazione
	9			Mostra M1
	10	Clicca sul bottone "Conferma accesso"		
	11			Mostra schermata Dashboard MC1

(La tabella prosegue sulla pagina successiva)

Estensione #a (L'Utente non autorizza la condivisione e torna indietro)	Step n.	Utente non registrato	API Google Cognito	Sistema
	6a	Annulla Autorizzazione di condivisione dati		
	7a		Torna alla schermata di accesso Google	
	8a	Preme pulsante Annulla e torna in M0		

2.5.2 Mockup: Cliente accede con e-mail Google

Mockup 0



Come per la registrazione, anche per l'accesso, a seguito della pressione del bottone "Google" si apre il browser predefinito che mostra la schermata per l'accesso con un account Google.

Una volta completati gli step del login con Google si ritorna su DietiEstate25 per completare la procedura di accesso mostrando il seguente mockup:

Alla pressione del bottone “Conferma accesso” si ha:

Mockup C1

2.5.2 Cockburn: Cliente accede con e-mail Google

USE CASE #4	Accedi con Google			
Obiettivo	Il Cliente riesce ad accedere a DietiEstates25			
Precondizioni	Il Cliente si trova sulla pagina iniziale di accesso dell'applicativo			
Condizione di successo	Il Cliente è riuscito a loggarsi con successo; Lo stesso si trova sulla Home dove può effettuare operazioni			
Condizione di fallimento	Il Cliente non riesce ad accedere all'applicativo			
Attore primario	Cliente			
Trigger	Il Cliente preme sul bottone "Google" in M0			
Main Scenario	**Step n.**	**Cliente**	**API Google Cognito**	**Sistema**
	1	Preme bottone "Google" in M0		
	2			Reindirizza l'Utente all'API Places
	3		Mostra schermata di accesso Google	
	4	Sceglie account e inserisce credenziali di Google		
	5		Convalida l'accesso	
	6		Genera token di accesso	
	7			Riceve token e completa l'accesso
	8			Mostra schermata Dashboard MC1

(La tabella prosegue sulla pagina successiva)

Estensione #a (Il Cliente non inserisce correttamente le credenziali Google)	**Step n.**	**Cliente**	**API Google Cognito**	**Sistema**
	4a	Non inserisce correttamente l'e-mail o il numero di telefono		
	5a	Preme il bottone "Avanti"		
	6a		Mostra il messaggio "Inserisci un indirizzo email o numero valido"	
	7a	Inserisce correttamente credenziali		
	8a	Preme il bottone "Avanti"		
	9a		Verifica l'identità	
	10a			Ritorna allo step 5 del Main Scenario

2.5.3 Mockup: Cliente ricerca immobili

Un cliente per poter effettuare una ricerca deve indicare una zona di interesse.
A patto che il cliente non scelga di modificare la tipologia di appartamento, di default, la ricerca avverrà su appartamenti in vendita.

Mockup C1

Una volta scritta la zona di interesse il cliente può premere sul pulsante che consente di avviare la ricerca:

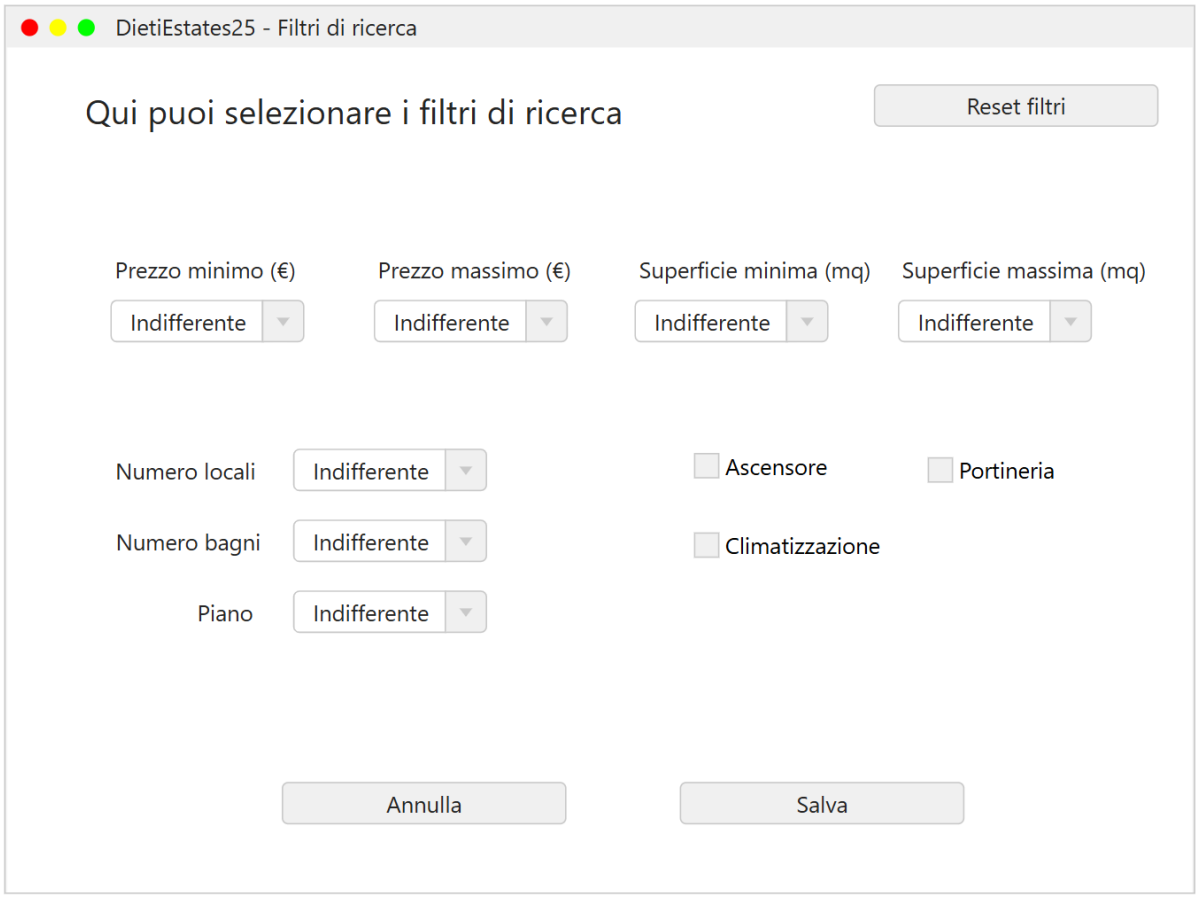
Mockup C2

Il mockup che segue mostra un possibile risultato di ricerca:

Mockup C3

Questo mockup invece mostra come apparirebbe all'utente la schermata per l'inserimento dei filtri di ricerca:

Mockup F



2.5.3 Cockburn: Cliente ricerca immobili

USE CASE #7	Ricerca Immobili		
Obiettivo	Il Cliente vuole effettuare una ricerca di immobili		
Precondizioni	Il Cliente ha effettuato l'accesso all'applicativo; Il Cliente si trova sulla Home MC1, dove può iniziare a fare una ricerca		
Condizione di successo	Il cliente, sulla base delle sue preferenze ed esigenze, riesce ad effettuare una ricerca di immobili		
Condizione di fallimento	Il Cliente non riesce ad effettuare la ricerca		
Attore primario	Cliente		
Trigger	Il Cliente preme sulla barra di ricerca		
Main Scenario	**Step n.**	**Cliente**	**Sistema**
	1	Preme sulla barra di ricerca e scrive la zona di interesse	
	2	Seleziona la tipologia di appartamento (vendita o affitto)	
	3	Preme sull'icona di ricerca	
	4		Mostra MC3 aggiornato con gli immobili trovati
Estensione #a (Il Cliente effettua una ricerca utilizzando i filtri)	**Step n.**	**Cliente**	**Sistema**
	3a	Preme sull'icona per le opzioni di ricerca	
	4a		Mostra MF (Mockup Filtri)
	5a	Seleziona opzioni di filtraggio degli immobili e preme "Salva"	
	6a		Aggiorna i filtri selezionati e mostra MC2
	7a		Ritorna allo step 3 del Main scenario

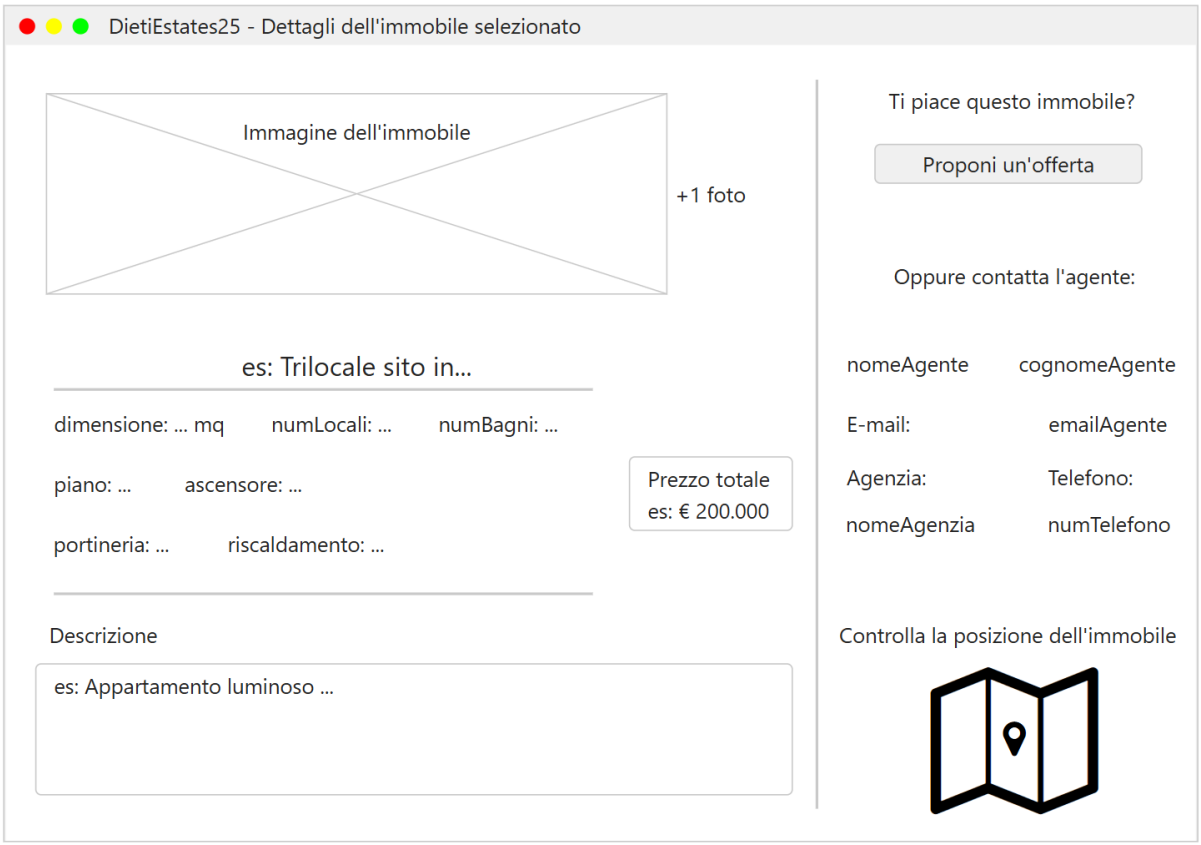
2.5.4 Mockup: Cliente propone un'offerta

Un cliente che ha effettuato l'accesso e ha avviato una ricerca, se ha individuato un immobile di suo gradimento ha la possibilità di proporre un'offerta.
Di seguito l'interfaccia che un cliente vede partendo da questa precondizione:

Mockup C1

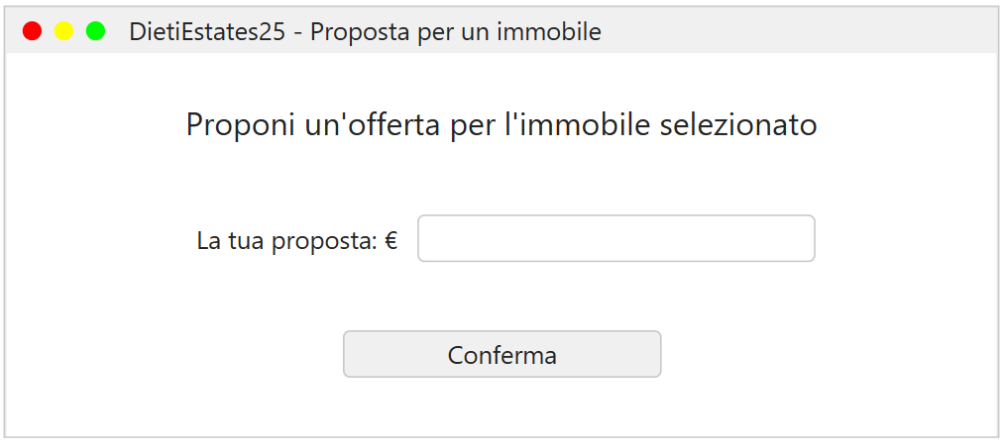
La selezione di uno degli immobili trovati consente la visualizzazione dei suoi dettagli, il successivo mockup mostra cosa vede il cliente dopo tale azione:

Mockup C2



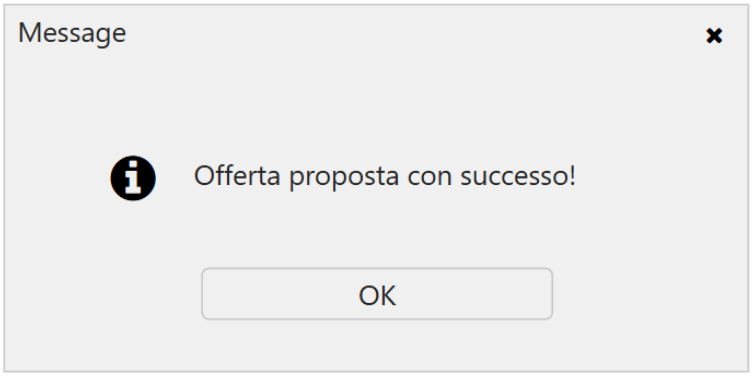
Il cliente cliccando sul bottone “Proponi un’offerta” ha la possibilità di mettersi in contatto con l’agente mostrando il suo interesse per l’immobile selezionato, di seguito il mockup che permette di inserire una proposta per l’immobile:

Mockup C3



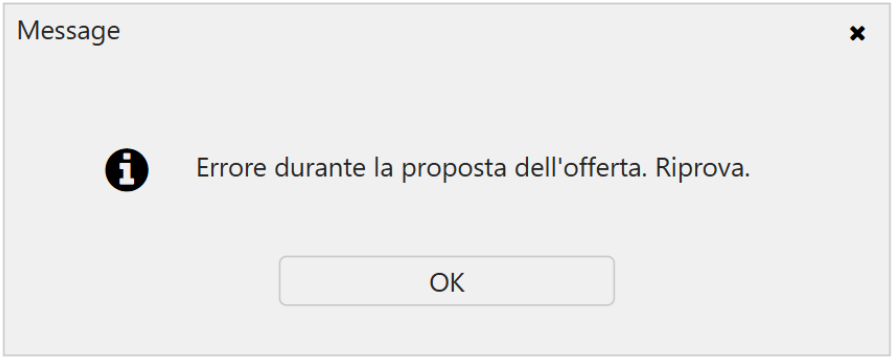
Se nel campo di testo viene inserita una proposta valida allora il cliente vedrà:

Mockup C4



In caso di proposta con valore numerico non valido il cliente vedrà:

Mockup C4



2.5.4 Cockburn: Cliente propone un'offerta

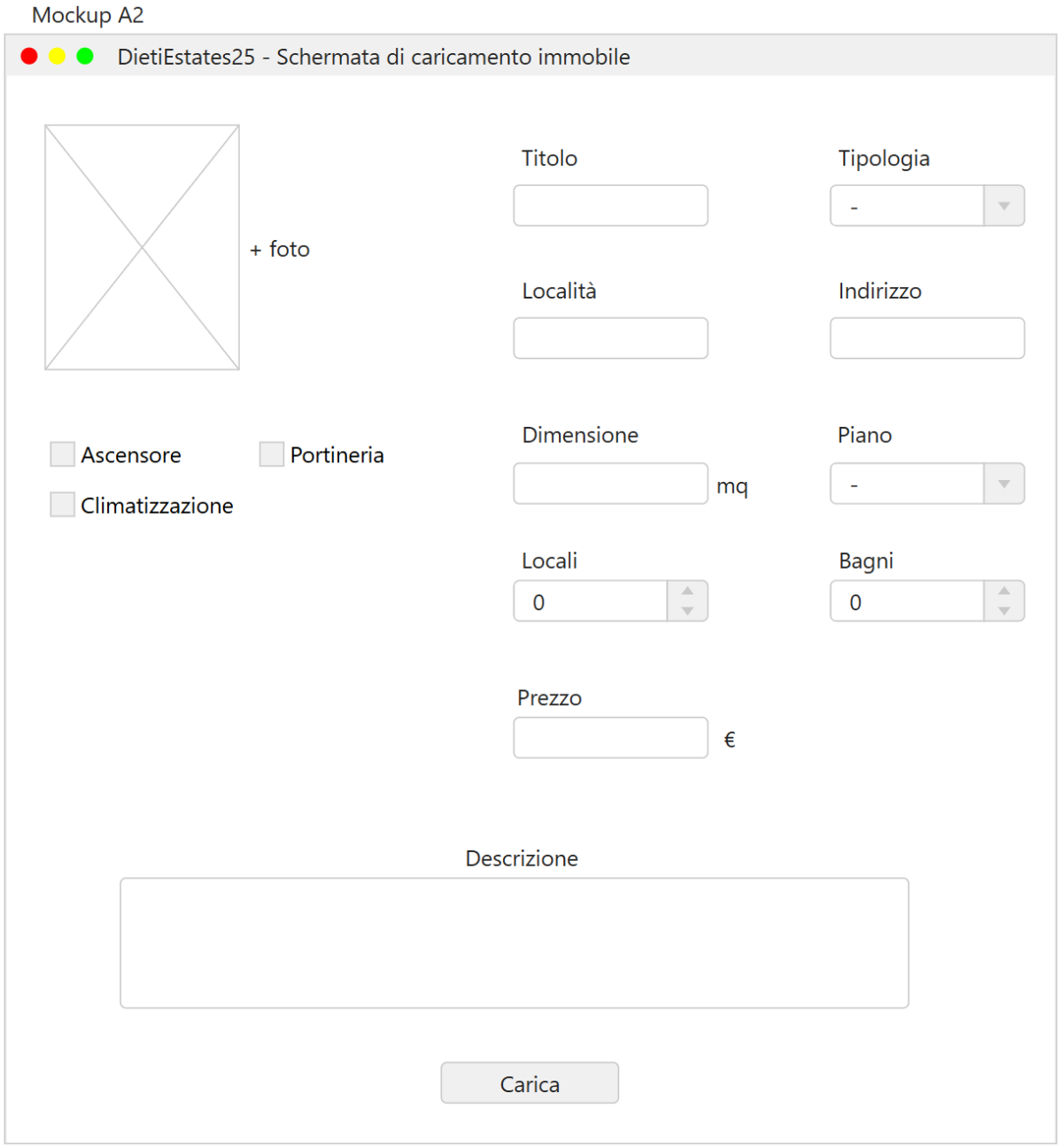
USE CASE #9	Proponi offerta		
Obiettivo	Il Cliente vuole proporre un'offerta per un immobile		
Precondizioni	Il Cliente ha effettuato l'accesso all'applicativo; Il Cliente si trova sulla Home MC1; Il Cliente ha già effettuato una ricerca di immobili e ne ha individuato uno di suo interesse		
Condizione di successo	Il Cliente riesce a comunicare all'agente immobiliare l'interesse per l'immobile riuscendo a proporre un'offerta		
Condizione di fallimento	Il Cliente non riesce a proporre un'offerta per l'immobile di interesse		
Attore primario	Cliente		
Trigger	Il Cliente preme sull'immobile di interesse		
Main Scenario	**Step n.**	**Cliente**	**Sistema**
	1	Preme sull'immobile di interesse	
	2		Mostra MC2
	3	Preme sul tasto "Proponi un'offerta"	
	4		Mostra MC3
	5	Inserisce nel campo di testo un'offerta valida per l'immobile	
	6	Preme il tasto "Conferma"	
	7		Mostra MC4 con scritto "Offerta proposta con successo!"
	8	Preme il tasto "Ok"	
	9		Ritorna su MC2
Estensione #a (Il Cliente propone un'offerta con un valore più alto di quello presentato dall'agente)	**Step n.**	**Cliente**	**Sistema**
	5a	Inserisce nel campo di testo un'offerta più alta di quella originale	
	6a	Preme il tasto "Conferma"	
	7a		Mostra MC4 con scritto "Errore durante la proposta dell'offerta. Riprova."
	8a	Preme il tasto "Ok"	
	9a		Ritorna su MC3
	10a		Riparti dallo step 5 del Main Scenario

2.5.5 Mockup: Agente immobiliare carica immobile

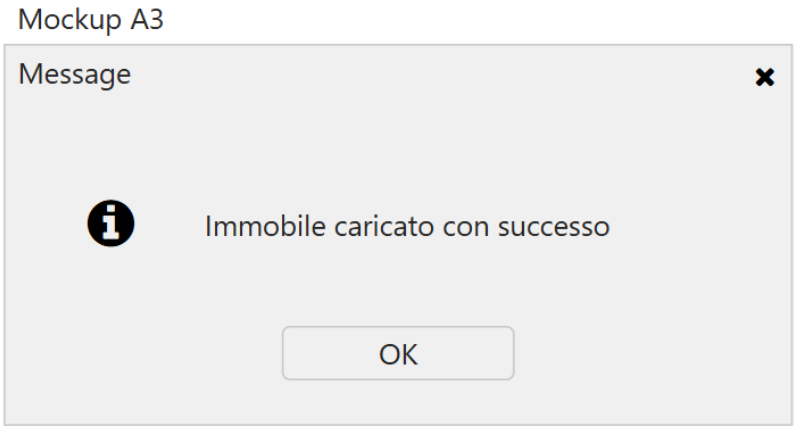
Schermata che un agente immobiliare vede dopo aver effettuato il login:

Mockup A1

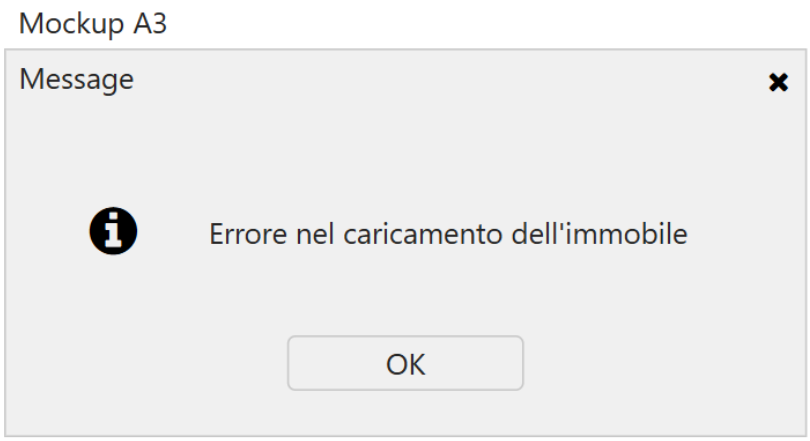
L’agente, una volta che ha cliccato sul bottone “Carica immobile”, si ritroverà un’interfaccia di questo tipo:



Quando l’agente avrà inserito correttamente tutti i dati si ritroverà davanti una schermata così:



Invece se il caricamento dell’immobile non avviene con successo verrà mostrata una schermata come quella che segue:



2.5.5 Cockburn: Agente immobiliare carica immobile

USE CASE #13	Carica Immobile		
Obiettivo	L'attore Agente Immobiliare vuole inserire un Immobile da vendere		
Precondizioni	L'Agente Immobiliare deve aver compiuto l'accesso		
Condizione di successo	L'Agente Immobiliare inserirà correttamente un Immobile visibile a tutti i Clienti		
Condizione di fallimento	L'Agente Immobiliare non inserirà un Immobile, Annullerà la conferma, Chiuderà l'applicazione durante il procedimento		
Attore primario	Agente Immobiliare		
Trigger	L'Agente Immobiliare preme pulsante "Carica immobile" in MA1		
Main Scenario	**Step n.**	**Agente Immobiliare**	**Sistema**
	1	Preme pulsante "Carica immobile" in MA1	
	2		Mostra schermata form in MA2
	3	Inserisce correttamente i dati obbligatori nel form e preme "Carica"	
	4		Mostra messaggio di successo in MA3
	5	Preme "OK" in MA3	
	6		Mostra dashboard agente MA1
Estensione #a (l'Agente Immobiliare non inserisce i dati correttamente)	**Step n.**	**Agente Immobiliare**	**Sistema**
	3a	Non inserisce correttamente i dati nel form e preme "Carica"	
	4a		Mostra messaggio di errore in MA3
Estensione #b (L'Agente Immobiliare annulla il riempimento del form)	**Step n.**	**Agente Immobiliare**	**Sistema**
	3b	Preme sulla "X" della finestra in MA2	
	4b		Mostra dashboard MA1

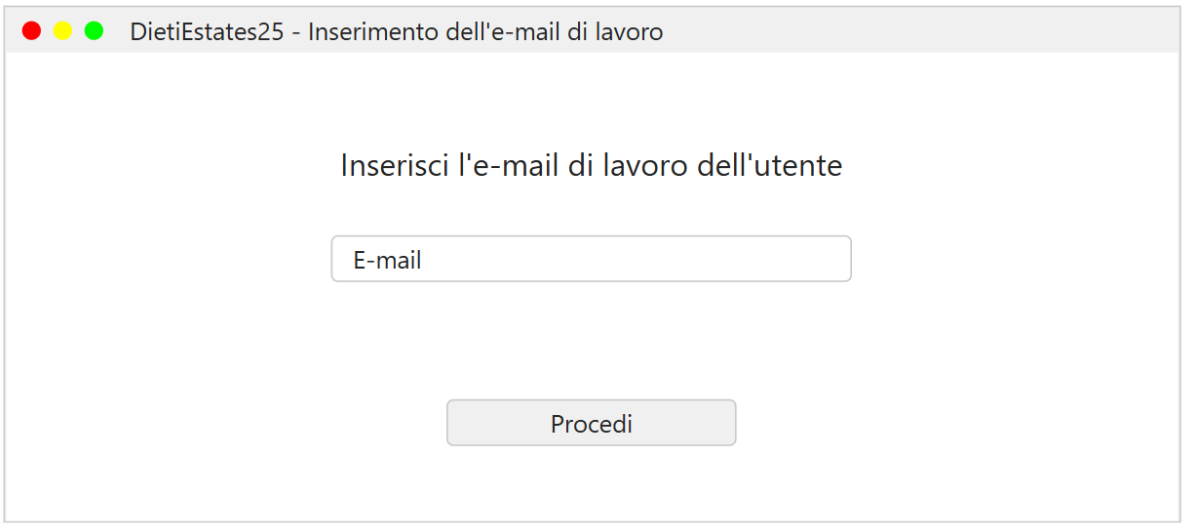
2.5.6 Mockup: Amministratore di supporto aggiunge un nuovo agente

Un amministratore di supporto, una volta che ha effettuato il login, avrà la seguente dashboard di amministrazione:

Mockup A1

Questa classe di utenti ha la possibilità di poter aggiungere un nuovo agente immobiliare.
La pressione del primo dei tre tasti (partendo da sinistra) in alto a destra nell'interfaccia, seguendo il menù, rimanda al mockup successivo:

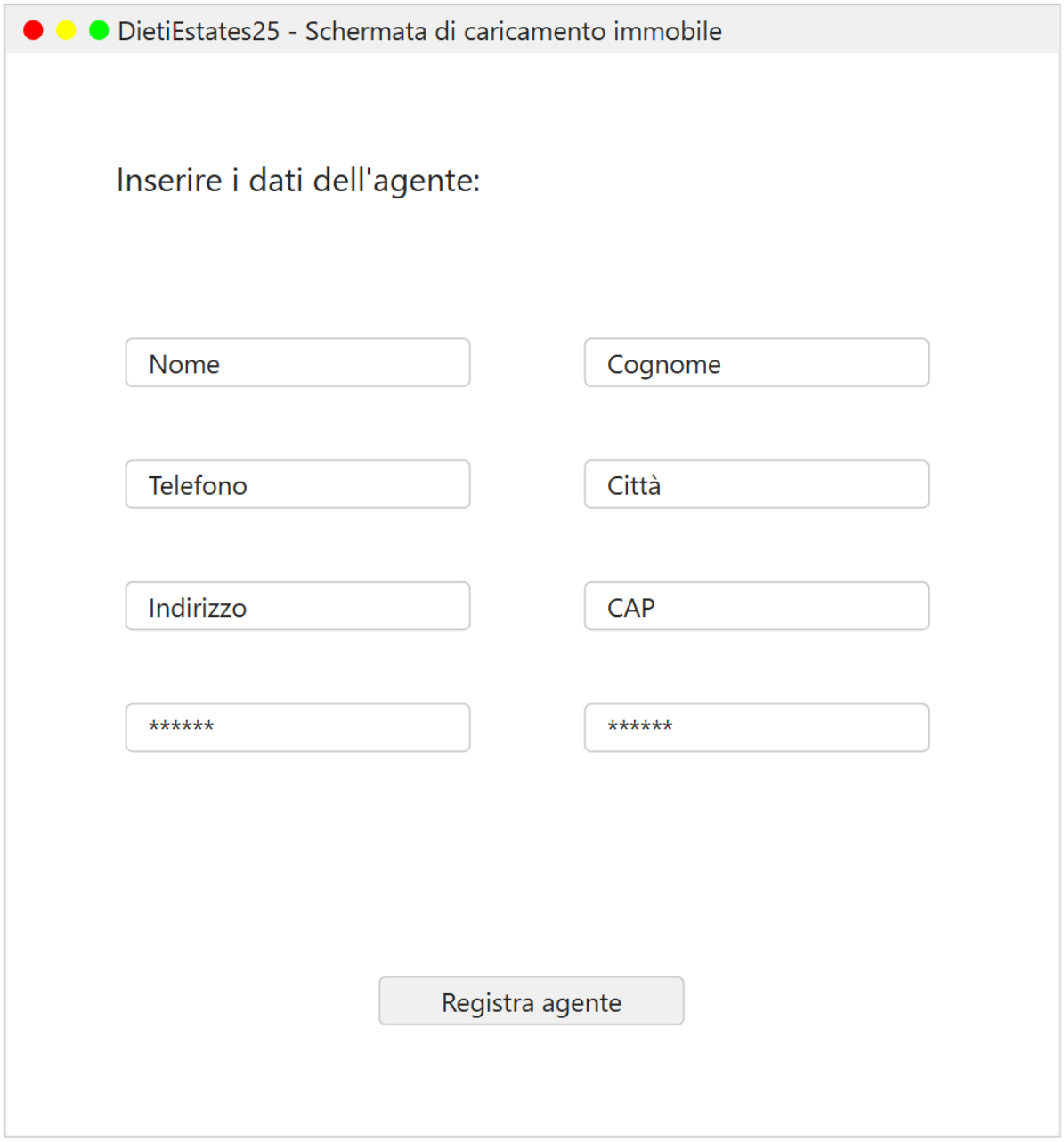
Mockup A2



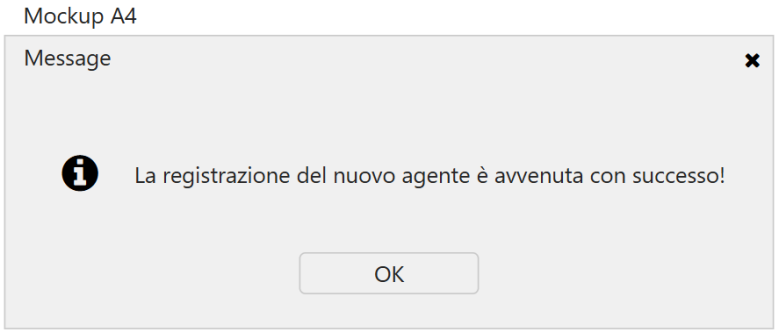
Una volta inserita l'e-mail di lavoro del nuovo agente si procede con l'inserimento dei dati del nuovo account.

Di seguito il mockup che illustra il form che l'amministratore di supporto dovrà compilare per completare la registrazione del nuovo agente immobiliare:

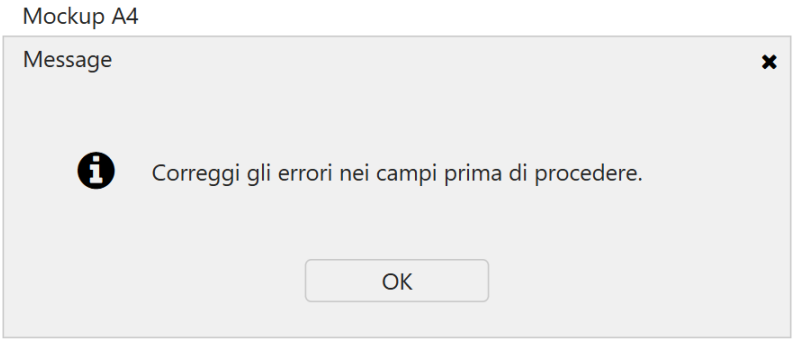
Mockup A3



La pressione del tasto “Registra agente”, in caso di successo, mostra il seguente mockup:



In caso di errore di validazione di uno o più input si avrà lo stesso mockup ma con un messaggio specifico riguardo l’errore:



2.5.6 Cockburn: Amministratore di supporto aggiunge un nuovo agente

USE CASE #16	Inserisci nuovo agente immobiliare		
Obiettivo	L'amministratore di supporto vuole aggiungere un nuovo agente immobiliare		
Precondizioni	L'amministratore di supporto ha effettuato l'accesso all'applicativo; L'amministratore di supporto si trova sulla dashboard MA1, dove può aggiungere un nuovo agente immobiliare		
Condizione di successo	L'amministratore di supporto riesce a portare a termine la successione di passi che dalla dashboard lo conducono all'inserimento del nuovo agente		
Condizione di fallimento	L'amministratore di supporto non riesce ad aggiungere un nuovo agente		
Attore primario	Amministratore di supporto		
Trigger	L'amministratore di supporto preme sul pulsante che consente di aggiungere nuovi agenti		
Main Scenario	**Step n.**	**Amministratore di supporto**	**Sistema**
	1	Preme sul pulsante per aggiungere nuovi agenti	
	2		Mostra il menu che permette di selezionare la voce relativa l'aggiunta di un nuovo agente
	3	Preme su "Aggiungi un nuovo agente immobiliare"	
	4		Mostra MA2 per inserire la nuova email
	5	Inserisce correttamente l'e-mail di lavoro e preme il tasto "Procedi"	
	6		Mostra MA3 per inserire i dati del nuovo agente
	7	Compila tutti i campi e preme il tasto "Registra agente"	
	8		Mostra MA4 con scritto "La registrazione del nuovo agente è avvenuta con successo!"
	9	Preme il tasto "Ok"	
	10		Mostra MA1
Estensione #a (L'amministratore di supporto non compila tutti i campi in fase di registrazione del nuovo agente)	**Step n.**	**Amministratore di supporto**	**Sistema**
	7a	Non compila tutti i campi e preme il tasto "Registra agente"	
	8a		Mostra MA4 con scritto "Correggi gli errori nei campi prima di procedere"
	9a	Preme il tasto "Ok"	
	10a		Ritorna al passo 7 del Main Scenario

3 Documento di Design del sistema

Il design del sistema rappresenta una fase cruciale nel ciclo di vita dello sviluppo software, in cui il modello concettuale derivato dall'analisi dei requisiti viene trasformato in un modello tecnico-operativo, pronto per l'implementazione.

Questo passaggio segna un cambio radicale di prospettiva: da un dialogo con il cliente, condotto in linguaggio naturale e orientato ai requisiti, in questa fase, si passa a una comunicazione strettamente tecnica col team di sviluppo, finalizzata a definire in modo chiaro e strutturato come il sistema dovrà essere realizzato.

Nelle sezioni successive del System Design sarà proposta e descritta l'architettura del sistema, la sua scomposizione in componenti più semplici (sottosistemi), e le strategie per soddisfare i requisiti non funzionali, come prestazioni, affidabilità, sicurezza e manutenibilità.
Il documento servirà da guida dettagliata per la successiva fase di sviluppo, garantendo coerenza, modularità e qualità durante l'implementazione.

Le attività cardine del System Design includono:

1. **Identificazione degli obiettivi di design**, derivati principalmente dai requisiti non funzionali e dalle priorità del progetto.
2. **Decomposizione del sistema in sottosistemi**, basandosi su use case e modelli di analisi, per ridurre la complessità e semplificarne lo sviluppo.
3. **Raffinamento della decomposizione**, attraverso l'applicazione di principi quali l'*alta coesione* e il *basso accoppiamento*, al fine di garantire indipendenza funzionale, facilità di manutenzione e riuso del codice.

Verranno approfonditi concetti fondamentali come il *Single Responsibility Principle* (ogni classe deve avere una sola responsabilità), la *Legge di Demetra* (per limitare le dipendenze tra oggetti) e il *Responsibility-Driven Design*, che orienta la progettazione verso una chiara attribuzione di responsabilità e collaborazione tra le classi. Attraverso esempi pratici (come la gestione dell'accesso ai dati o l'interazione tra oggetti in scenari reali) saranno illustrate le conseguenze delle scelte progettuali sulla robustezza, flessibilità e manutenibilità del sistema.

3.1 Descrizione dell'architettura proposta

L'architettura di DietiEstates25 segue il pattern MVC (Model-View-Controller), esso consente di separare le responsabilità logica di business (Model), la presentazione dell'interfaccia utente (View) e la gestione degli input utente (Controller):

- **Presentation Layer:** Esso prevede tutte quelle classi la cui interfaccia utente è realizzata con Java Swing.
- **Business Logic Layer:** Questo livello gestione regole di dominio e coordinamento.
- **Data Access Layer:** Astrazione della persistenza con l'ausilio del pattern DAO.

Questa separazione garantisce:

- **Manutenibilità:** Le modifiche ad uno strato non impattano gli altri.
- **Testabilità:** Ogni componente può essere agevolmente testato.
- **Scalabilità:** La possibilità di sostituire componenti senza riscrivere l'intera applicazione.

3.2 Scelte tecnologiche adottate

In questa sezione si spiegano le scelte tecnologiche adottate per la realizzazione del back-end, dell'infrastruttura e dei servizi impiegati.

3.2.1 Back-End

- **Linguaggio**: Java 21. Scelto per la sua maturità, ampia diffusione nel settore enterprise e supporto nativo alla programmazione object-oriented, requisito fondamentale della traccia.
- **Build Tool**: Apache Maven. Utilizzato per la gestione automatizzata delle dipendenze, del ciclo di build e per l'integrazione con strumenti di qualità come SonarQube.
- **Database**: PostgreSQL 42.5.3, gestito tramite PgAdmin. Scelto per la sua affidabilità, supporto a tipi di dati avanzati (es. JSON) e per essere un database relazionale open-source robusto.

3.2.2 Infrastruttura & Servizi

- **Cloud Provider**: Amazon Web Services (AWS). Sfruttiamo i seguenti servizi:
  a) **RDS (Relational Database Service)**: Per ospitare il database PostgreSQL in modo gestito, garantendo backup automatici e alta disponibilità.

b) **Cognito**: Per delegare la gestione sicura di autenticazione, registrazione e profili utente.

- **Strumenti di Sviluppo**:
  a) **IDE**: Eclipse. Utilizzato per lo sviluppo del back-end Java.
  b) **Analisi Codice**: SonarQube. Integrato nel processo di build per analisi statica continua della qualità e sicurezza del codice.

3.3 Schema della persistenza dei dati

Il modello di persistenza di DietiEstates25 è stato ricavato dal Modello di Dominio e raffinato attraverso l'applicazione dei principi di normalizzazione e delle best practice per l'integrità referenziale. Lo schema logico-relazionale descritto in questa sezione è implementato fisicamente su *PostgreSQL* in ambiente *AWS RDS*.

La struttura è rappresentata più avanti, nella sezione 3.3.5, essa mostra il diagramma delle classi di persistenza dove ciascuna classe corrisponde a una tabella del database relazionale.

3.3.1 Gerarchia degli account e autenticazione

La gestione degli utenti è modellata attraverso una gerarchia di ereditarietà.

- **Account**: tabella base contenente i dati comuni a tutti gli utenti del sistema.
  - Attributi: idAccount (PK), email, name, cognome, telefono, password, citta, indirizzo, cap, ruolo.
  - Il campo ruolo (String) discrimina il tipo di utente (Cliente, Agente, Admin, Supporto).
- **Cliente**: estende Account. Aggiunge l'attributo idCliente (PK, FK su Account.idAccount) e la lista delle offerte iniziali proposte.
- **AgenteImmobiliare**: estende Account. Aggiunge idAgente (PK), agenzia (String) e la lista delle risposte alle offerte (risposteOfferte).
- **AmministratoreDiSupporto**: estende Account. Aggiunge idSupporto (PK, FK).
- **AmministratoreAgenzia**: estende Account. Aggiunge idAdmin (PK, FK).

Scelta progettuale: L'autenticazione è delegata ad **AWS Cognito** per la gestione sicura delle credenziali. La tabella Account mantiene una replica dei soli dati anagrafici (nome, cognome, email, ruolo) necessari per le operazioni di join e profilazione, garantendo così l'indipendenza dal provider di autenticazione

3.3.2 Modello immobiliare

- **Immobile**: entità centrale del dominio.
    - Attributi: idImmobile (PK), foto (List<Byte>), titolo, indirizzo, località, dimensione, descrizione, tipologia.
    - Chiave esterna: agenteAssociato (FK su AgenteImmobiliare.idAgente).
    - Relazione: un Agente può caricare molti Immobili (1 a N); un Immobile è gestito da un solo Agente.

Per le sottoclassi di Immobile invece:

- **ImmobileInVendita**: estende Immobile. Aggiunge l'attributo prezzoTotale.
- **ImmobileInAffitto**: estende Immobile. Aggiunge l'attributo prezzoMensile.

Filtri:

- Attributi: idFiltri (PK), prezzoMinimo, prezzoMassimo, superficieMinima, superficieMassima, piano, numeroLocali, numeroBagni, ascensore, portineria, climatizzazione.

3.3.3 Gestione delle offerte e controproposte

Il sottosistema delle offerte implementa un meccanismo di tracciamento bidirezionale tra cliente e agente.

- **OffertaIniziale**: rappresenta la proposta economica avanzata da un cliente.
    - Attributi: idOfferta (PK), importoProposto, dataOfferta, stato (in attesa, accettata, rifiutata, controproposta).
    - Chiavi esterne: clienteAssociato (FK su Cliente.idCliente), immobileAssociato (FK su Immobile.idImmobile).
    - Relazione: un Cliente può proporre molte Offerte (1 a N); un Immobile può ricevere molte Offerte (1 a N).
- **RispostaOfferta**: modella la reazione dell'agente all'offerta del cliente.
    - Attributi: idRisposta (PK), tipoRisposta (accettata, rifiutata, controproposta), controproposta (double), dataRisposta, attiva (boolean).
    - Chiavi esterne: offertaInizialeAssociata (FK su OffertaIniziale.idOfferta), agenteAssociato (FK su AgenteImmobiliare.idAgente).
    - Relazione: 1 a 1 tra OffertaIniziale e RispostaOfferta attiva. Il flag *attiva* consente di gestire storicamente più risposte nel tempo (es. ciclo controproposta >> nuova offerta >> nuova risposta), mantenendo tracciato l'intero iter negoziale.

3.3.4 Scelte tecnologiche per la persistenza

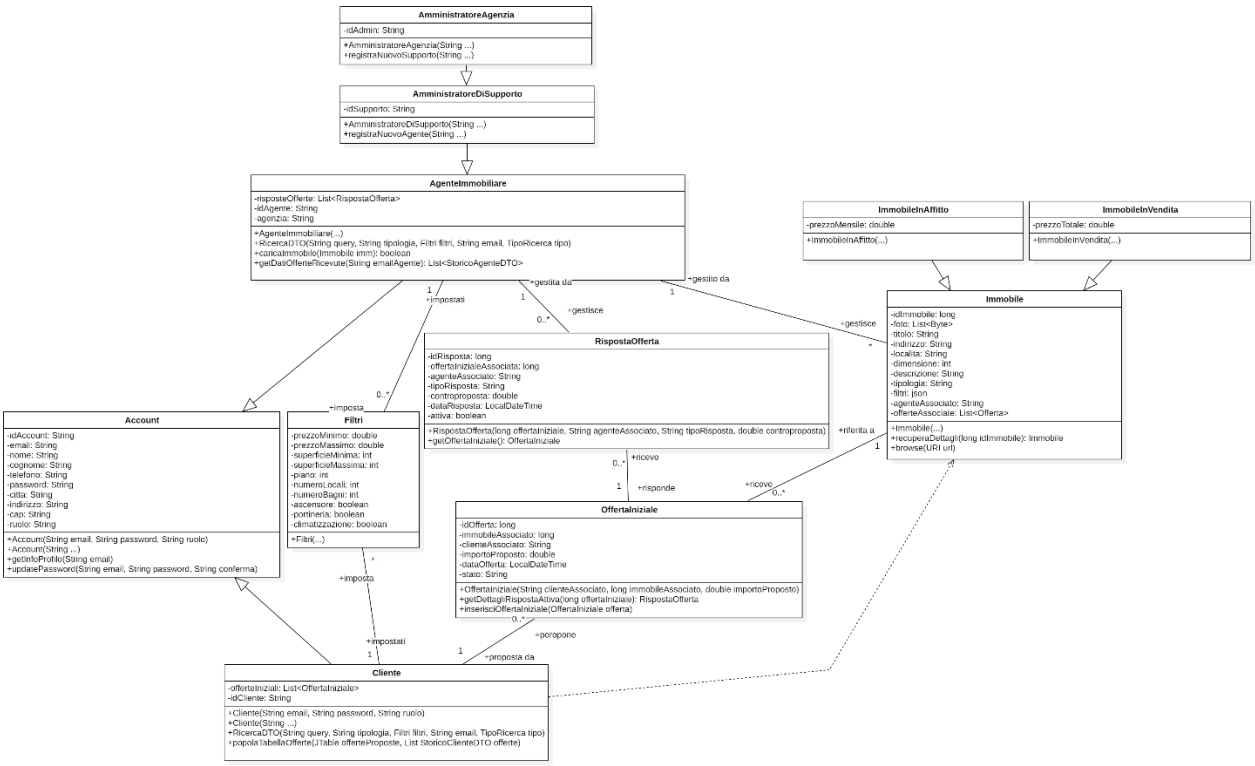
Il database è ospitato su *AWS RDS* con motore *PostgreSQL*. La scelta è motivata da tre requisiti fondamentali:

- **Affidabilità e disponibilità:** AWS RDS garantisce backup automatici e alta disponibilità soddisfacendo i requisiti non funzionali di robustezza del sistema.
- **Integrità transazionale:** Il rispetto dei vincoli ACID è essenziale per operazioni critiche quali la creazione contestuale di un'offerta e della relativa risposta, o la prenotazione di una visita in concorrenza.

L'accesso ai dati è incapsulato attraverso il *pattern DAO*, che separa la logica di business dalla tecnologia di persistenza, garantendo manutenibilità e testabilità.

3.3.5 UML

Il diagramma UML che segue è stato realizzato col tool “StarUML”:



3.4 Convenzioni di design per l’interfaccia utente

L'interfaccia utente, realizzata con Java Swing, è stata progettata seguendo i principi di usabilità di Nielsen e le linee guida della piattaforma desktop per garantire un'esperienza familiare e prevedibile.

3.4.1 Coerenza e Standard

- **Layout:** Ogni interfaccia suggerisce visivamente un’area di interazione iniziale che consente all’utente di esprimere input verso il sistema (generalmente posta in alto o sulla destra dell’interfaccia) e un’area di output dei risultati (posta nella parte inferiore dell’interfaccia o in tutta l’area se l’interfaccia non richiede input utente).
- **Nomenclatura:** I pulsanti per le azioni principali seguono una nomenclatura standard: "Cerca", "Salva", "Carica Immobile", "Proponi Offerta". I colori sono utilizzati con parsimonia per non affaticare l'utente (es. blu per un’azione che porta ad un successo, rosso per errore o annullamento di un’operazione).

3.4.2 Controllo e Libertà dell'Utente

- Ogni finestra per l'inserimento di dati (Es.: "Carica Immobile") è provvista di pulsanti "Annulla" (alle volte, in assenza di esso, è sostituita dalla "X") per la chiusura, permettendo all'utente di uscire dall'operazione senza conseguenze.
- Ogni finestra dispone di un tasto (generalmente quello più utilizzato. Es.: tasti del tipo “Prosegui” o “Cerca”) predisposto come default alla pressione del tasto Enter della tastiera.
- Per qualsiasi categoria di utente, nelle interfacce come quella relativa l’accesso o la dashboard, la digitazione dei caratteri della tastiera compare direttamente nel campo di maggiore interesse (Es.: campo di testo per la ricerca di un immobile)

3.4.3 Feedback e Prevenzione degli Errori

- **Feedback:** Ogni azione dell'utente (caricamento di un immobile, visualizzazione dello stato di valutazione di un’offerta iniziale) produce un feedback visivo immediato. Vengono utilizzati messaggi in dialog (finestre informative) di conferma.
- **Prevenzione:** I campi di input non consentono di inserire dati non validi (Es.: campo di testo in un campo numerico)

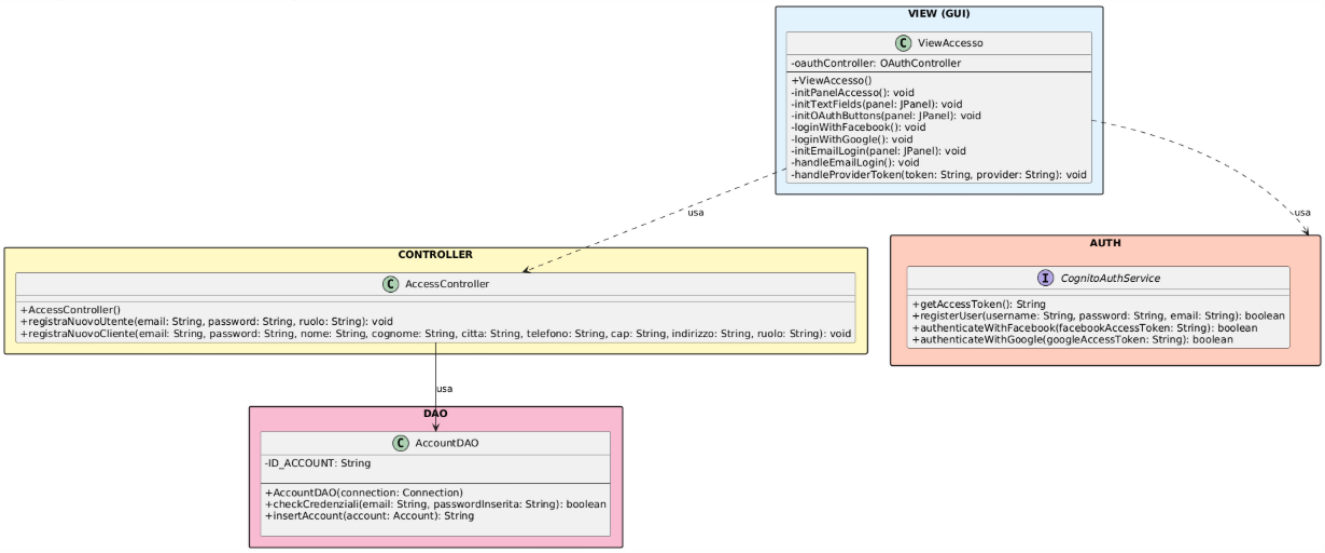
3.5 Diagramma delle classi di design

I diagrammi presentati in questa sezione rappresentano la struttura statica del sistema, modellata con PlantUML. Essi aderiscono al pattern architetturale MVC integrato con il pattern DAO per la persistenza. Per ogni funzionalità, viene evidenziata la suddivisione delle responsabilità:

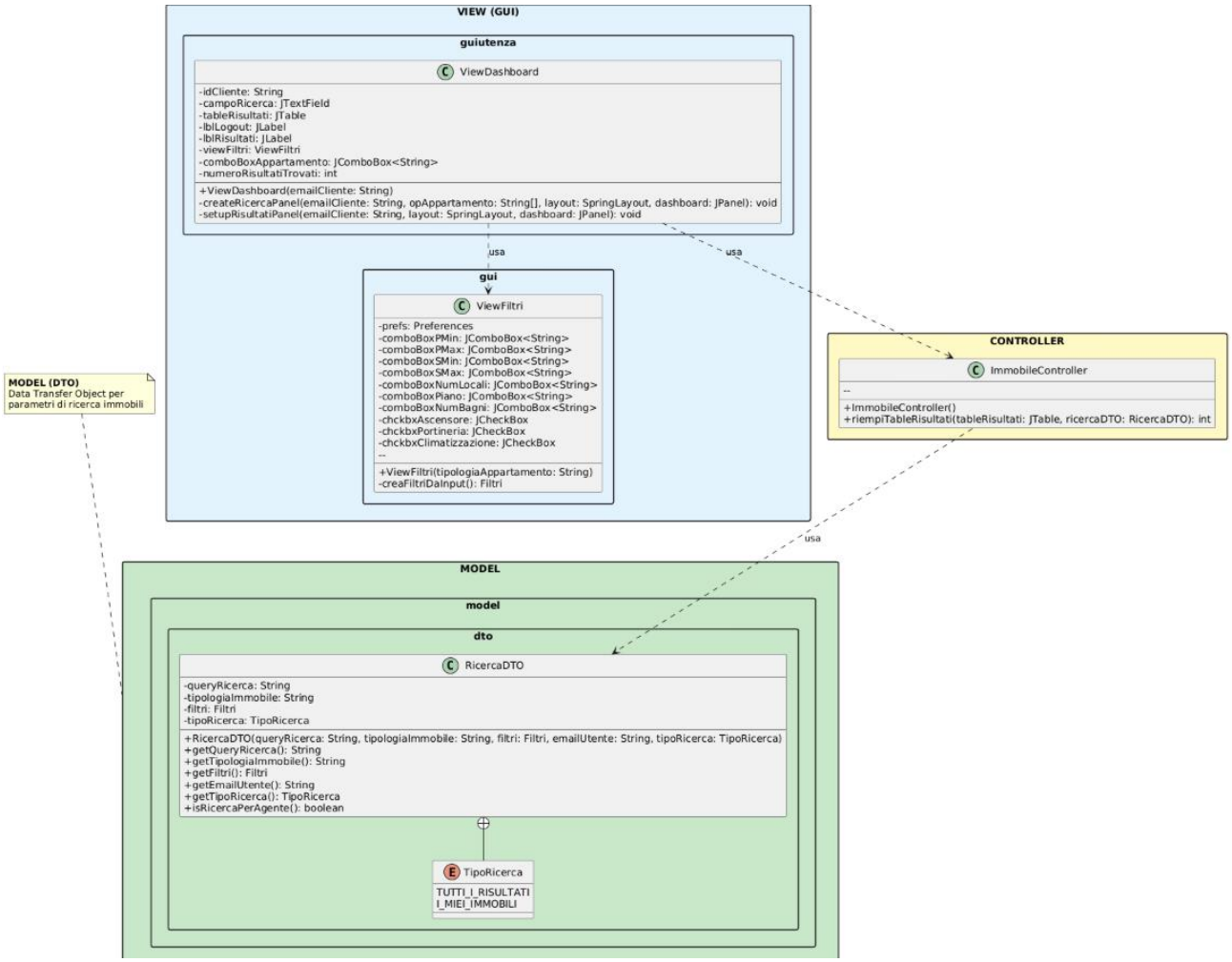
- **View:** Classi Swing che gestiscono la presentazione e catturano gli eventi utente.
- **Controller:** Gestiscono la logica di applicazione, ricevono gli input dalla View, interrogano il model e aggiornano la View.
- **Model:** Contengono la logica di business e le regole di dominio. Operano su interfacce DAO.
- **DAO:** Incapsulano la logica di accesso ai dati (CRUD) sul database PostgreSQL, isolando il livello dei servizi dalle specifiche implementative di persistenza.

Il tool impiegato per la realizzazione dei diagrammi che seguono si chiama “PlantUML Web Server”.

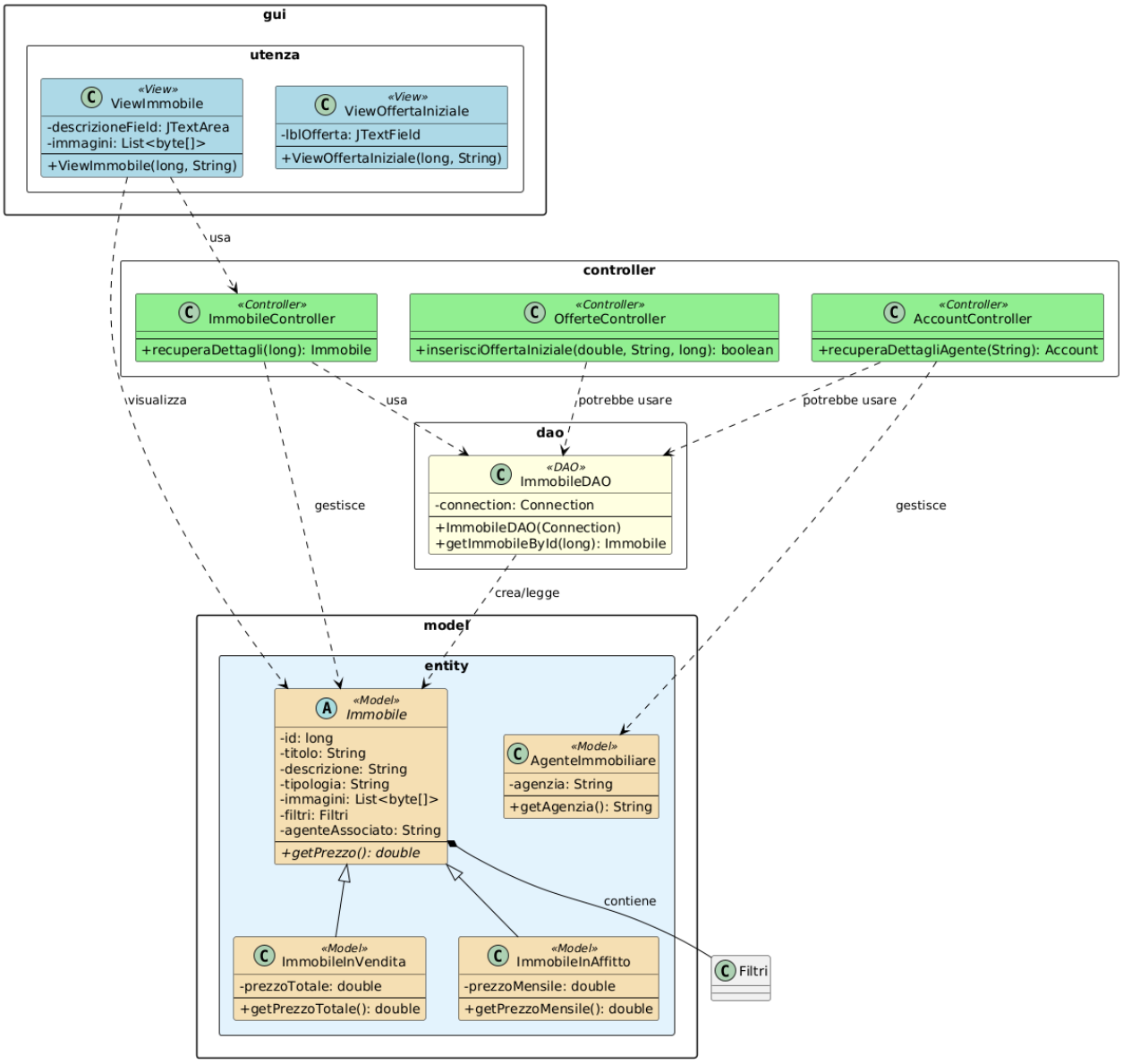
3.5.1 Registrazione mediante e-mail / Facebook / Google



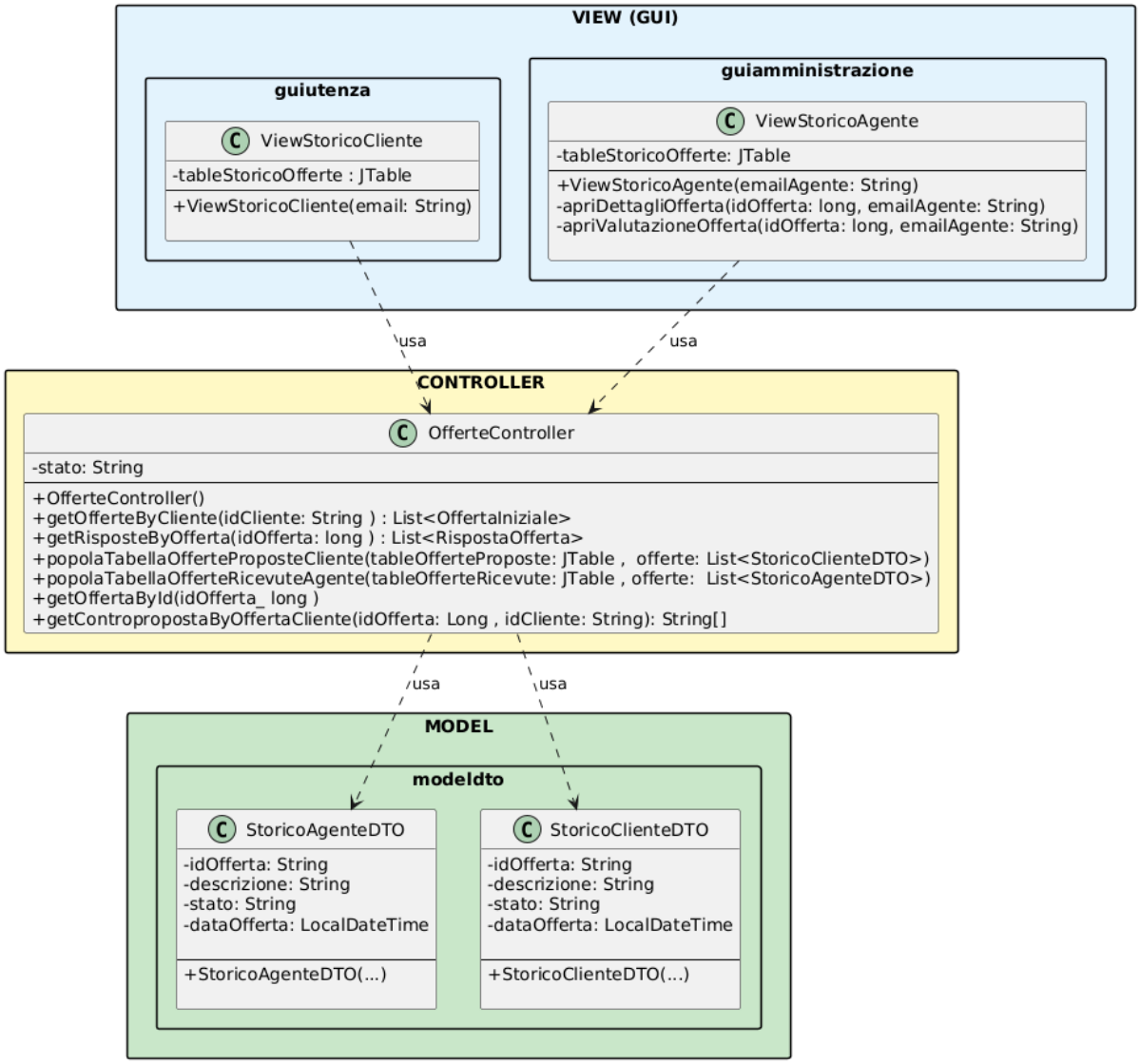
3.5.2 Cliente ricerca un immobile



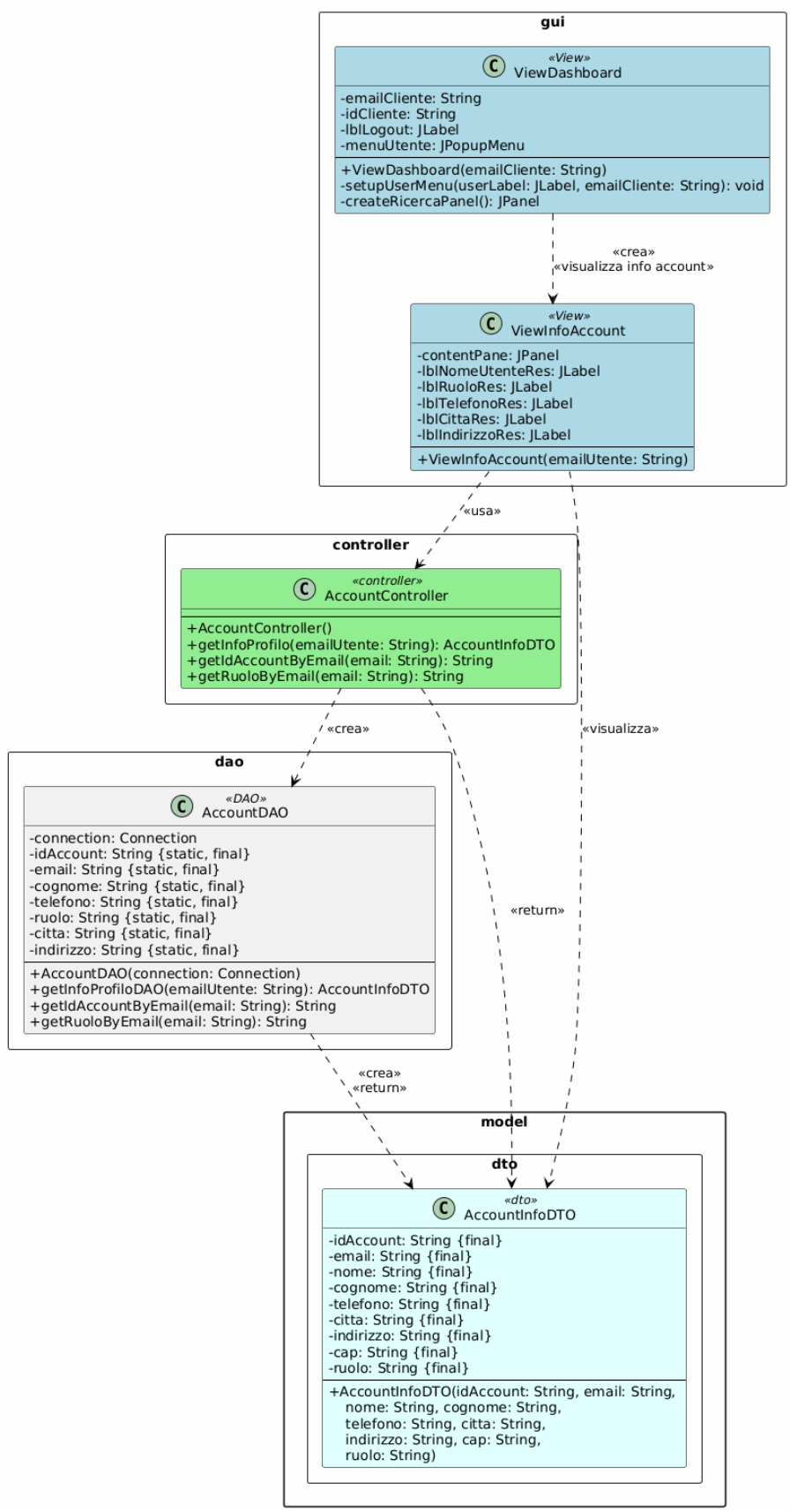
3.5.3 Cliente propone un’offerta



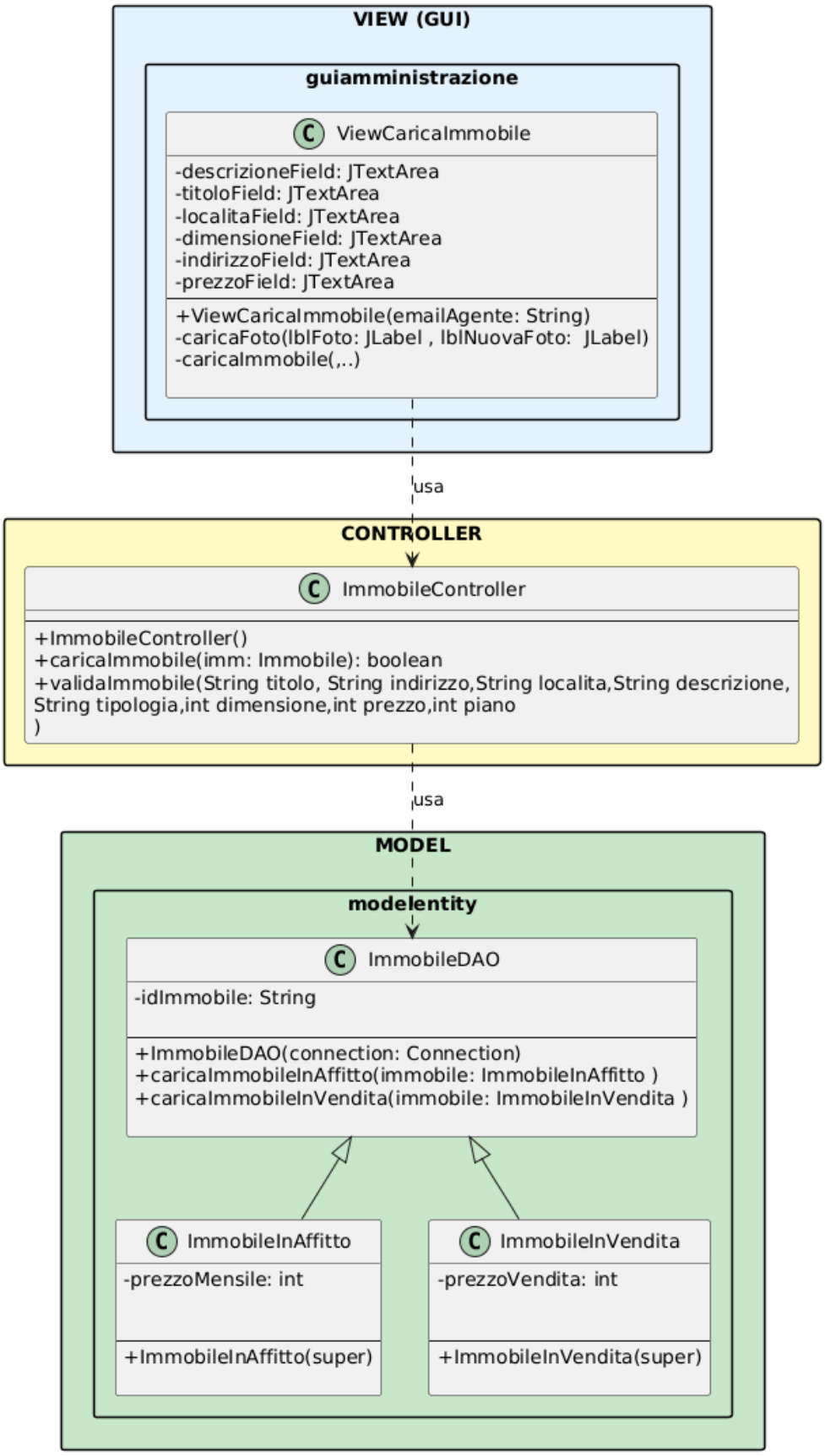
3.5.4 Storico cliente e agente



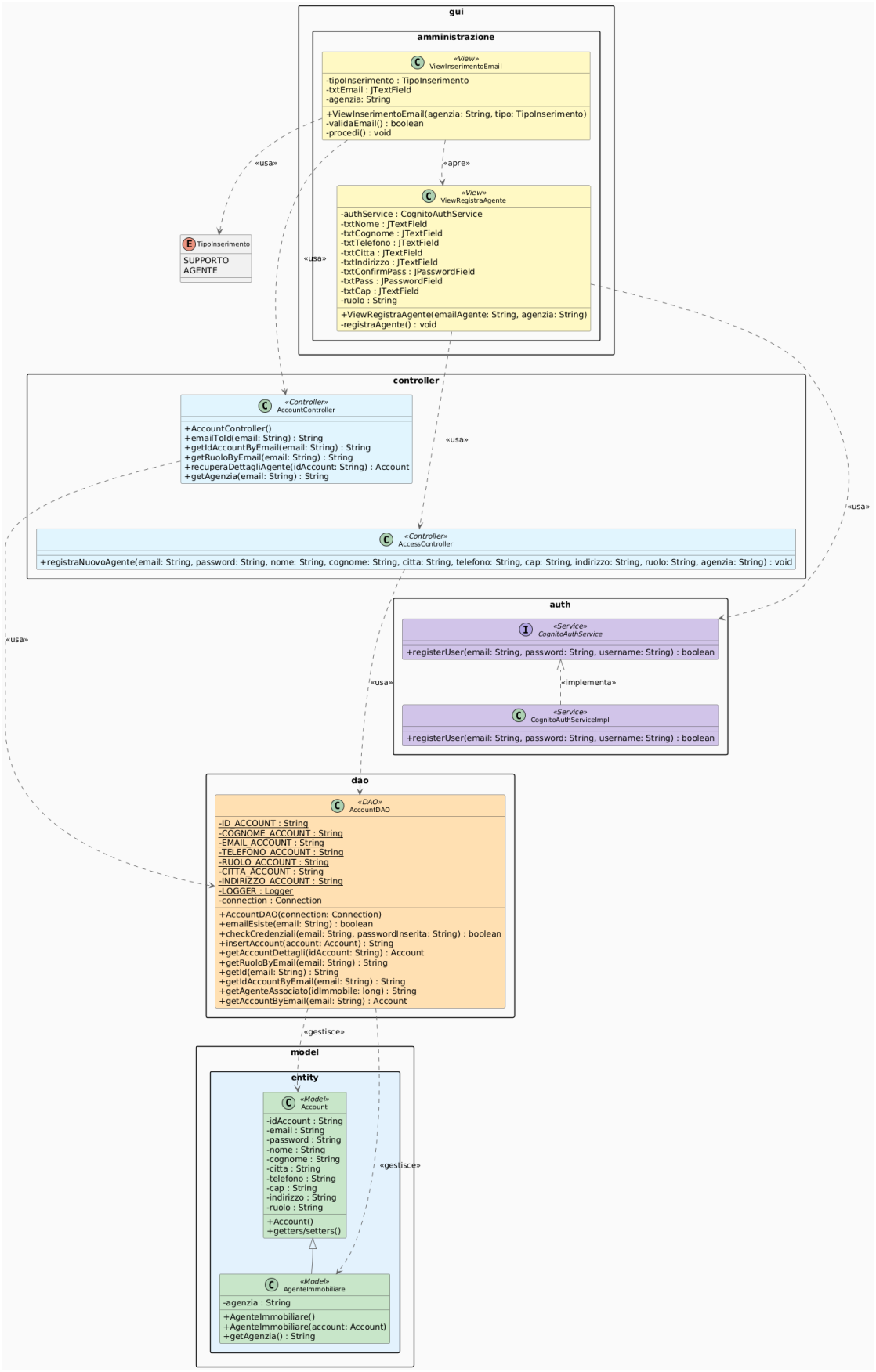
3.5.5 Cliente visualizza le informazioni del proprio account



3.5.6 Agente immobiliare carica un immobile



3.5.7 Amministratore di supporto aggiunge un nuovo agente



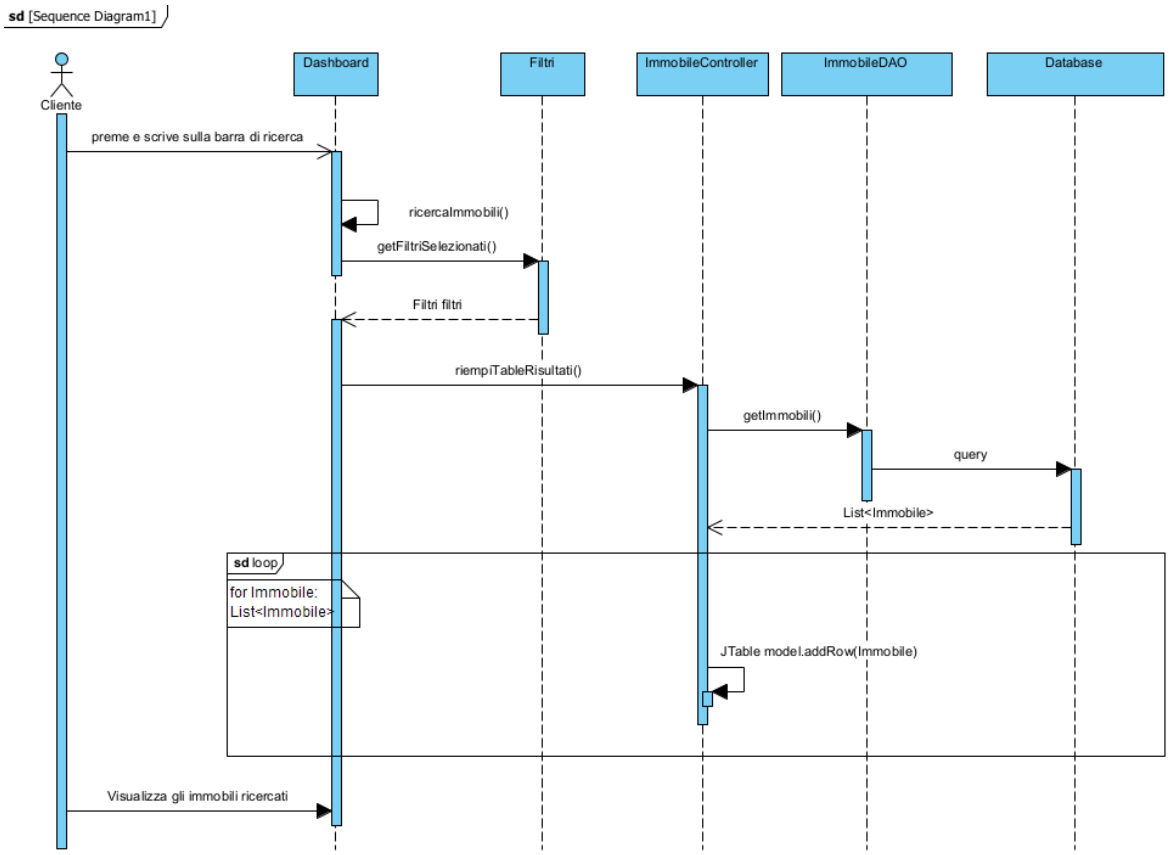
3.6 Diagrammi di sequenza di design

I diagrammi di sequenza sono un tipo di diagramma comportamentale nel linguaggio di modellazione UML, esso rappresenta l'interazione dinamica tra gli oggetti in un sistema. Il loro scopo principale è visualizzare il flusso di messaggi scambiati nel tempo per eseguire una specifica funzionalità o scenario d'uso.
Di seguito verranno mostrati i comportamenti dei casi d'uso (per ordine) 7, 9, 13 e 16 mediante una rappresentazione dinamica.

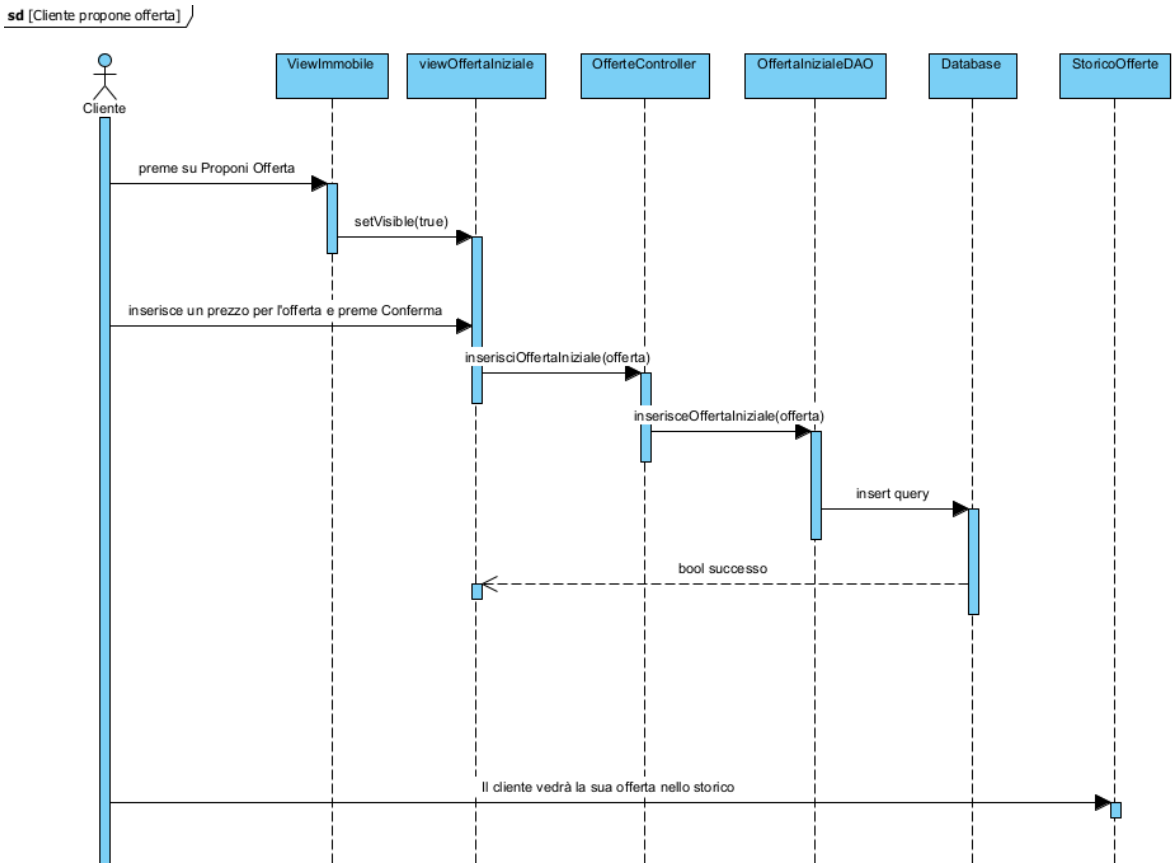
Il tool impiegato per la realizzazione dei seguenti sequence diagram è "Visual Paradigm".

(vedi pagina seguente)

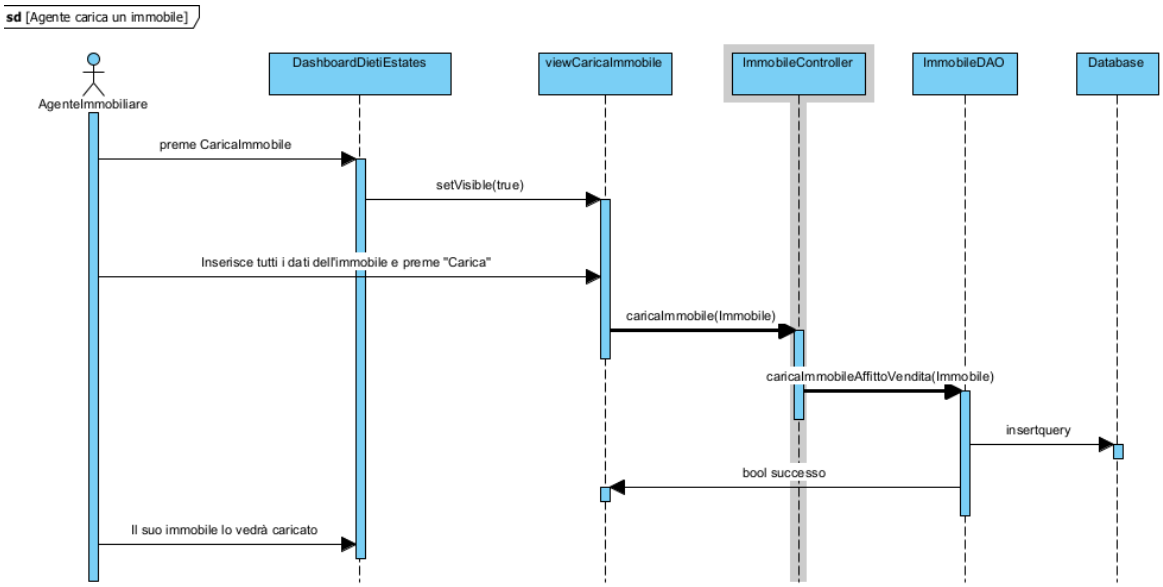
3.6.1 Sequence diagram: Cliente ricerca immobili



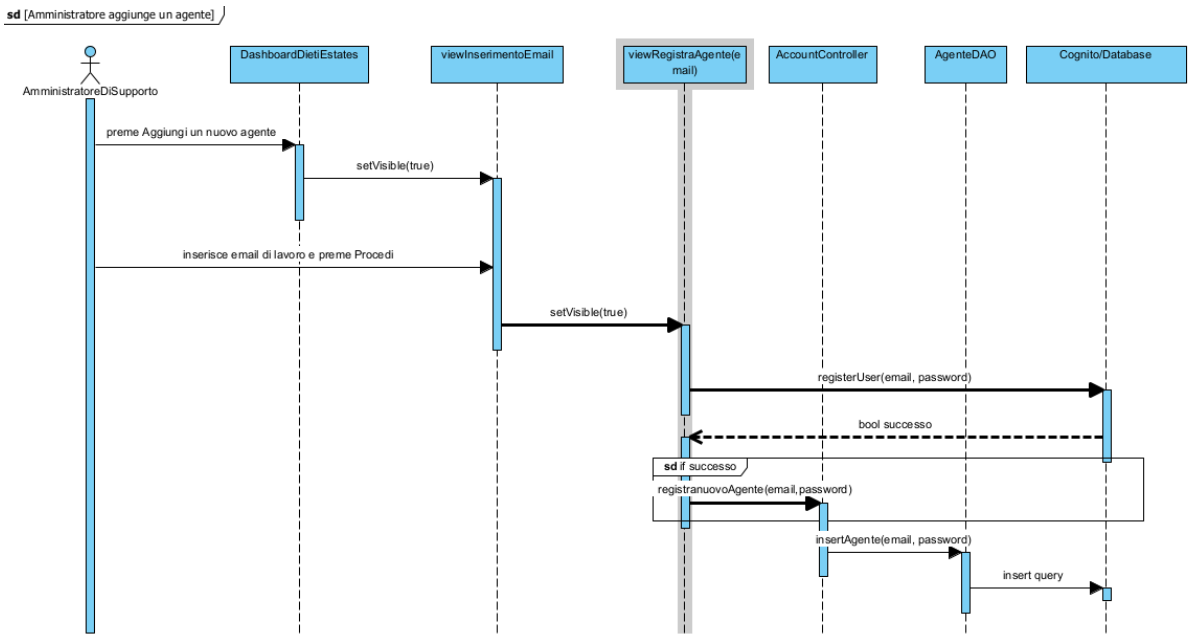
3.6.2 Sequence diagram: Cliente propone un’offerta



3.6.3 Sequence diagram: Agente immobiliare carica immobile



3.6.4 Sequence diagram: Amministratore di supporto aggiunge agente



4 Documento sul processo di sviluppo

Questo capitolo descrive gli strumenti e le metodologie adottate per la gestione del ciclo di vita del software, dalla compilazione automatica al controllo di versione, fino alla garanzia della qualità del codice. L'obiettivo è dimostrare l'adozione di pratiche ottimali al fine di garantire l'ampliamento di funzionalità e la manutenibilità del prodotto.

4.1 Illustrazione del file di build automatica

Per automatizzare il processo di compilazione, gestione delle dipendenze e packaging dell'applicazione, è stato adottato *Apache Maven*. L'uso di Maven garantisce la riproducibilità della build su qualsiasi macchina e l'integrazione con l'ambiente di integrazione continua (CI) e gli strumenti di analisi statica.

Contenuto del Dockerfile:

```
# Build con Maven
FROM maven:3.9-eclipse-temurin-21 AS build
WORKDIR /app
COPY pom.xml .
COPY src ./src
RUN mvn clean package -DskipTests

# Runtime
FROM eclipse-temurin:21-jre
WORKDIR /app
COPY --from=build /app/target/DietiEstates25.jar app.jar

# Avvio applicazione
ENTRYPOINT ["java", "-jar", "app.jar"]
```

Altri dettagli sono specificati all'interno del file README.

4.2 Strumenti di versioning impiegati

Il controllo di versione è stato gestito attraverso il sistema distribuito *Git*, con repository ospitato su *GitHub*. L'intero flusso di lavoro è stato supportato dall'uso di *GitHub Desktop* come interfaccia grafica, garantendo un basso overhead operativo e una curva di apprendimento rapida per ambo i membri del team.

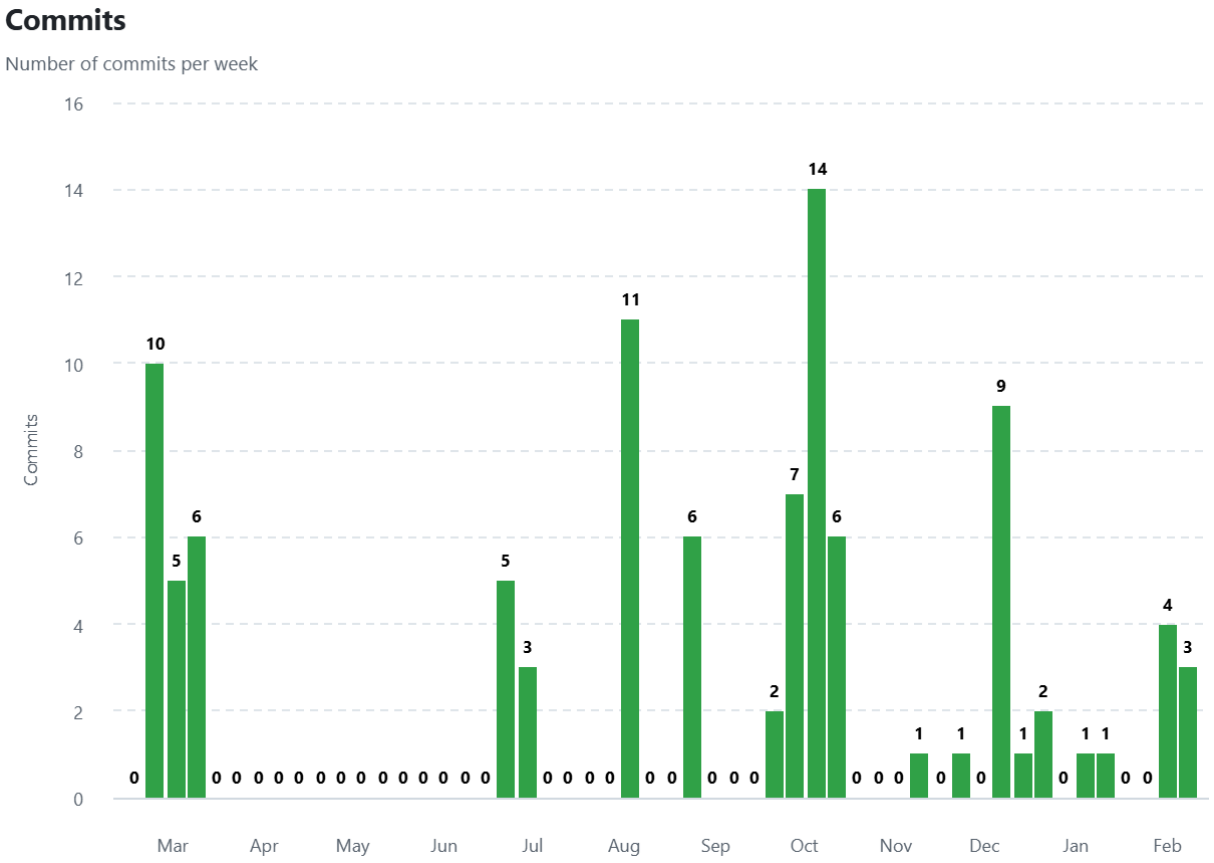
Strategia di Branching:
È stato adottato un modello di branching semplificato sintetizzato dal seguente elenco:

- Il branch main contiene sempre codice stabile e pronto per il rilascio.
- Al termine dello sviluppo e del testing locale, il branch viene integrato in main tramite *Pull Request*.
- Ogni commit è sempre stato chiaramente documentato da una descrizione generale (Summary) e da un'altra più dettagliata.

Statistiche di Repository:

- **Totale Commit:** ~100
- **Contributors:** 2 (Visconti Gaetano, Borgia Manuele)
- **Frequenza Commit:** Attività pressoché distribuita su tutto l'arco del progetto con picchi nelle fasi iniziali e nella fase di ultimazione del codice.
- **Primo commit:** 23/02/2025
- **Ultimo commit:** 12/02/2026

L’immagine che segue riporta l’istogramma delle statistiche sopra elencate:



L’immagine è stata recuperata dalla sezione *Insights* del repository che ospita il codice.

4.3 Report della qualità del codice con SonarQube

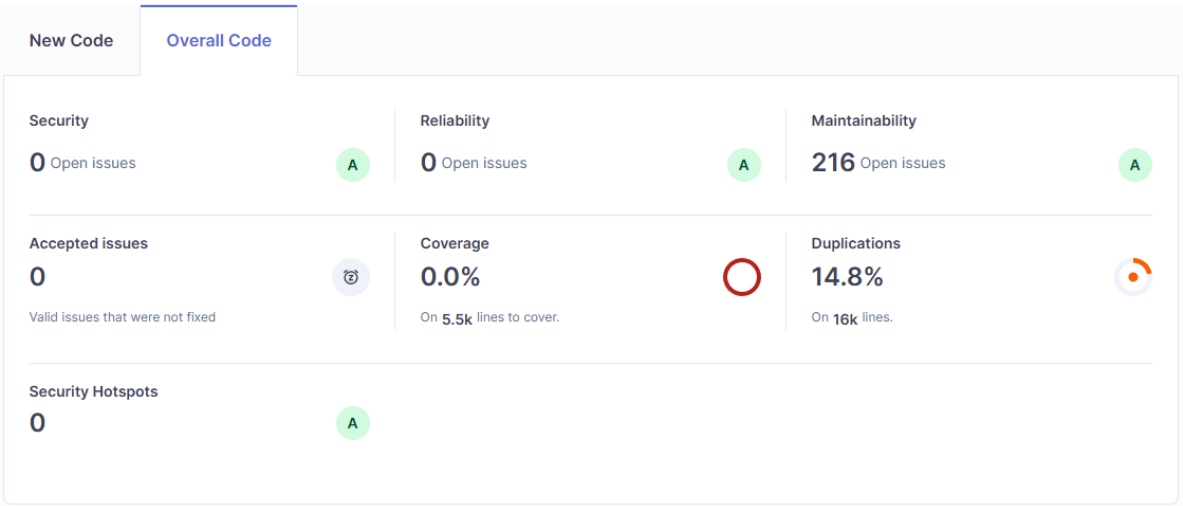
Per poter rilasciare un buon prodotto software bisogna garantire che il codice scritto sia un codice di qualità. A tal fine, è stato integrato nel processo di sviluppo la piattaforma *SonarQube*, eseguita localmente ed integrata con Maven.

SonarQube ha permesso di eseguire un'analisi statica continua del codice, identificando e classificando problemi secondo tre criteri principali: *Bug, Vulnerabilità e Code Smell*. L'analisi è stata limitata al modulo back-end.

Strategia di Analisi:

- **SonarLint in IDE:** Durante la scrittura del codice, SonarLint per Eclipse ha fornito feedback in tempo reale, suggerendo correzioni immediate su code smells e potenziali bug.
- **Analisi On-Demand con Maven:** Prima del merge, è stato lanciato il comando *mvn clean verify sonar:sonar* per generare un report completo.

Report dell'ultima analisi:
L'ultima analisi eseguita sul branch main ha prodotto i seguenti risultati:



L'utilizzo combinato di SonarLint e SonarQube ha garantito il mantenimento di un livello di qualità del codice costantemente elevato, riducendo il debito tecnico e prevenendo l'introduzione di vulnerabilità.

5 Documento sul testing e sulla valutazione dell'usabilità

Arrivati in questa fase della progettazione del prodotto software si rende necessario testare quanto è stato realizzato, nonostante la disponibilità di strumenti di *Automated Unit Testing* è impossibile testare completamente un modulo che non sia non banale, del resto *"Il testing può solo dimostrare la presenza di bug, non la loro assenza"* (Dijkstra).

Il presente capitolo e le sue sezioni descrivono le strategie adottate per la progettazione di quattro casi di test che di seguito verranno presentati e descritti.
I test sono stati implementati utilizzando il framework *JUnit 5 (xUnit)* e hanno l'obiettivo di verificare il corretto comportamento di metodi non banali caratterizzati da almeno due parametri di input.
Infine, vi è un'altra sezione che concentra la sua attenzione sull'aspetto dell'*usabilità* intessa come comprensibilità del sistema da parte dell'utente.

5.1 Unit testing e strategie adottate

I criteri di progettazione del prodotto software adottati includono:

- Black-box testing con partizionamento in classi di equivalenza.
- Analisi dei casi limite.

A seguire, nel dettaglio, i criteri sopra citati:

Col *black-box testing* la definizione dei casi di test è fondata sui requisiti funzionali dell'unità da testare, il sistema viene testato concentrandosi sul dominio di input del programma e questo è possibile grazie al *partizionamento in classi di equivalenza* che consiste nel scegliere insiemi finiti di valori da domini di input (potenzialmente infiniti).
Per ogni classe verranno selezionati uno o pochi casi di test rappresentativi, così facendo si riduce il numero totale di test necessari mantenendo altresì una buona (ma non ottimale) copertura funzionale, ciò è possibile applicando la strategia N-WECT.

L'analisi dei casi limite, invece, si concentra sui valori estremi degli intervalli di input, come i valori minimi, massimi e quelli immediatamente vicini. Questa tecnica è efficace perché gli errori si verificano spesso proprio ai confini degli intervalli validi. Testare i casi limite permette quindi di individuare *bugs*.

5.1.1 Test: isValidEmail (String email, boolean allowEmpty)

Il metodo verifica la validità di un indirizzo e-mail sulla base:

- del contenuto della stringa `email.`
- del valore booleano `allowEmpty`, che indica se una stringa vuota è ammessa.

Sono state individuate le seguenti classi di equivalenza:

Parametro `email:`

Classe	Valore
CE1	`email == null`
CE2	`email` stringa vuota
CE3	`email` con formato valido
CE4	`email` con formato non valido

Parametro `allowEmpty:`

Classe	Valore
CE5	`allowEmpty = true`
CE6	`allowEmpty = false`

I casi di test sono stati selezionati in modo da coprire tutte le classi di equivalenza rilevanti e le principali combinazioni tra i parametri:

- CE1: e-mail nulla
- CE2 + CE5: e-mail vuota consentita
- CE3 + CE6: e-mail con formato corretto
- CE4 + CE6: e-mail con formato errato

Elenco dei test xUnit:

```java
@Test
void emailNulla_nonValida() {
    // Arrange / Let
    String email = null;

    // Act
    boolean result = controller.isValidEmail(email, false);

    // Assert
    assertFalse(result);
}

@Test
void emailVuotaConsentita() {
    // Arrange / Let
    String email = "";

    // Act
    boolean result = controller.isValidEmail(email, true);

    // Assert
    assertTrue(result);
}

@Test
void emailFormatoCorretto_valida() {
    // Arrange / Let
    String email = "utente@mail.com";

    // Act
    boolean result = controller.isValidEmail(email, false);

    // Assert
    assertTrue(result);
}

@Test
void emailFormatoErrato_nonValida() {
    // Arrange / Let
    String email = "user@ma il.com";

    // Act
    boolean result = controller.isValidEmail(email, false);

    // Assert
    assertFalse(result);
}
```

5.1.2 Test: isValidPassword (String password, String conferma)

Il metodo verifica la validità di una password confrontando:

- La stringa `password`
- La stringa `conferma`

Sono state individuate le seguenti classi di equivalenza:

Classe	Valore
CE1	`password == null` e `conferma == null`
CE2	`password` diversa da `conferma`
CE3	`password` uguale a `conferma` e valida
CE4	`password` uguale a `conferma` ma non conforme alle regole

Elenco dei test xUnit:

```java
@Test
void passwordNulla_nonValida() {
    // Arrange / Let
    String password = null;
    String conferma = null;

    // Act
    boolean result = controller.isValidPassword(password, conferma);

    // Assert
    assertFalse(result);
}

@Test
void passwordDiversaDaConferma_nonValida() {
    // Arrange / Let
    String password = "abc123";
    String conferma = "abc124";

    // Act
    boolean result = controller.isValidPassword(password, conferma);

    // Assert
    assertFalse(result);
}

@Test
void passwordValida() {
    // Arrange / Let
    String password = "abc123";
    String conferma = "abc123";

    // Act
    boolean result = controller.isValidPassword(password, conferma);

    // Assert
    assertTrue(result);
}

@Test
void passwordSenzaNumeri_nonValida() {
    // Arrange / Let
    String password = "abcdef";
    String conferma = "abcdef";

    // Act
    boolean result = controller.isValidPassword(password, conferma);

    // Assert
    assertFalse(result);
}
```

5.1.3 Test: validaImmobile(...)

Il metodo `validaImmobile()` riceve in ingresso un insieme eterogeneo di parametri testuali e numerici che descrivono un immobile e verifica la loro validità. In caso di input non valido, il metodo segnala l'errore mediante il lancio di una `IllegalArgumentException`;
in caso contrario, l'esecuzione termina senza eccezioni.

Classe	Metodo	Valore
CE1	testInputValido()	Campi testuali non vuoti, tipologia definita, dimensione>0, prezzo>0, piano≥-1
CE2	testCampoTestualeVuoto()	Titolo vuoto ("")
CE3	testTipologiaNonValida()	Tipologia pari a "-"
CE4	testValoriNumericiNonValidi()	Dimensione=0
CE5	testPianoNonValido()	Piano=-5

Elenco dei test xUnit:

```
@Test
void testInputValido() {
    assertDoesNotThrow(() ->
        controller.validaImmobile(
                "Casa",
                "Via Roma 10",
                "Milano",
                "Appartamento luminoso",
                "Vendita",
                80,
                200000,
                2
        )
    );
}

/**
 * Classe di equivalenza: CAMPI TESTUALI NON VALIDI
 * Almeno un campo testuale vuoto
 */
@Test
void testCampoTestualeVuoto() {
    assertThrows(IllegalArgumentException.class, () ->
        controller.validaImmobile(
                "",                 // titolo non valido
                "Via Roma 10",
                "Milano",
                "Descrizione",
                "Affitto",
                70,
                1000,
                1
        )
    );
}

/**
 * Classe di equivalenza: TIPOLOGIA NON VALIDA
 */
@Test
void testTipologiaNonValida() {
    assertThrows(IllegalArgumentException.class, () ->
        controller.validaImmobile(
                "Casa",
                "Via Roma",
                "Milano",
                "Descrizione",
                "-",                // tipologia non valida
                70,
                1000,
                1
        )
    );
}
```

```java
@Test
void testValoriNumericiNonValidi() {
    assertThrows(IllegalArgumentException.class, () ->
        controller.validaImmobile(
                "Casa",
                "Via Roma",
                "Milano",
                "Descrizione",
                "Vendita",
                0,                // dimensione non valida
                1000,
                1
        )
    );
}

/**
 * Classe di equivalenza: PIANO NON VALIDO
 * Piano < -1
 */
@Test
void testPianoNonValido() {
    assertThrows(IllegalArgumentException.class, () ->
        controller.validaImmobile(
                "Casa",
                "Via Roma",
                "Milano",
                "Descrizione",
                "Vendita",
                80,
                1000,
                -5               // piano non valido
        )
    );
}
```

5.1.4 Test: isValidControproposta (double controproposta, double offertaIniziale)

Il metodo verifica la validità di una controproposta economica confrontandola con un'offerta iniziale.

Il valore di ritorno è booleano:

- `true` se la controproposta è valida
- `false` in caso contrario

Classe	Metodo	Valore
CE1	contropropostaValida()	Controproposta = 1200 Offerta iniziale = 1000
CE2	contropropostaMinoreOffertaIniziale_nonValida()	Controproposta = 800 Offerta iniziale = 1000
CE3	contropropostaUgualeOffertaIniziale_nonValida()	Controproposta = 1000 Offerta iniziale = 1000
CE4	contropropostaNegativa_nonValida()	Controproposta = −500
CE5	contropropostaZero_nonValida()	Controproposta = 0

Elenco dei test xUnit:

```
@Test
void contropropostaValida() {
    // Arrange / Let
    double controproposta = 1200.0;
    double offertaIniziale = 1000.0;

    // Act
    boolean result = controller.isValidControproposta(controproposta, offertaIniziale);

    // Assert
    assertTrue(result);
}

@Test
void contropropostaMinoreOffertaIniziale_nonValida() {
    // Arrange / Let
    double controproposta = 800.0;
    double offertaIniziale = 1000.0;

    // Act
    boolean result = controller.isValidControproposta(controproposta, offertaIniziale);

    // Assert
    assertFalse(result);
}

@Test
void contropropostaUgualeOffertaIniziale_nonValida() {
    // Arrange / Let
    double controproposta = 1000.0;
    double offertaIniziale = 1000.0;

    // Act
    boolean result = controller.isValidControproposta(controproposta, offertaIniziale);

    // Assert
    assertFalse(result);
}

@Test
void contropropostaNegativa_nonValida() {
    // Arrange / Let
    double controproposta = -500.0;
    double offertaIniziale = 1000.0;

    // Act
    boolean result = controller.isValidControproposta(controproposta, offertaIniziale);

    // Assert
    assertFalse(result);
}
```

```java
@Test
void contropropostaZero_nonValida() {
    // Arrange / Let
    double controproposta = 0;
    double offertaIniziale = 1000.0;

    // Act
    boolean result = controller.isValidControproposta(controproposta, offertaIniziale);

    // Assert
    assertFalse(result);
}
```

5.2 Valutazione dell'usabilità

L'usabilità è il grado con cui un sistema può essere utilizzato da utenti specifici per raggiungere obiettivi definiti in modo efficace, efficiente e soddisfacente. Jakob Nielsen, uno dei principali studiosi del tema, ha formalizzato questo concetto nel suo libro *Usability Engineering* (1993), che è diventato un riferimento fondamentale nel campo dell'interazione uomo-macchina.

Secondo Nielsen, l'usabilità si basa su *cinque attributi di qualità*:

- **Apprendibilità**: quanto è facile per un nuovo utente imparare a usare il sistema.
- **Efficienza**: quanto rapidamente l'utente riesce a svolgere i compiti una volta appreso l'uso del sistema.
- **Memorabilità**: quanto è facile ricordare come usare il sistema dopo un periodo di inattività.
- **Errori**: numero e gravità degli errori commessi dagli utenti e facilità di recupero.
- **Soddisfazione**: quanto l'esperienza d'uso risulta piacevole e gratificante.

5.2.1 Expert reviews / checklist

L'Expert Review è una tecnica di valutazione dell'usabilità in cui si analizza un sistema software per identificare problemi di utilizzo e punti di forza. Lo scopo è garantire che l'applicazione sia facile da usare, coerente e intuitiva prima di sottoporla a test con utenti finali.

In questo progetto la review è stata condotta utilizzando una checklist basata sulle principali euristiche di usabilità, adattata al contesto di un'applicazione per clienti e agenti immobiliari. La checklist permette di valutare in maniera sistematica aspetti quali *feedback, coerenza, controllo dell'utente, prevenzione degli errori, linguaggio ed efficienza delle operazioni*.

Categoria	Criterio	Descrizione	Risultato
Feedback	Stato Offerte	L'utente visualizza offerte chiaramente dopo aver proposto un prezzo per un immobile	Molto chiaro
	Stato annuncio	L'agente vede chiaramente se un annuncio è in attesa, valutato o controproposta	Molto chiaro
Coerenza	Pulsanti uniformi	Tutti i pulsanti simili hanno colore, etichetta e posizione coerenti	Pulsanti coerenti
	Layout pagine	Struttura delle pagine simili per funzioni simili (es. Lista immobili, dettagli immobile)	Strutture molto simili
Controllo e libertà	Carica annuncio	L'agente può caricare un annuncio facilmente senza errori	Caricamento semplice
	Annulla operazioni	L'utente e l'agente possono annullare o tornare indietro facilmente in caso di errore	Torna indietro semplice
Prevenzione errori	Form di inserimento	I campi obbligatori sono evidenziati e gli errori segnalati prima del salvataggio	Errori segnalati ed evidenziati
	Input numerici	Prezzo, metratura, numero stanze, ecc. richiedono input numerici validi	Input sempre validi o evidenziati
Linguaggio	Termini comprensibili	Tutti i termini sono chiari agli utenti	Termini molto chiari
	Messaggi di errore	I messaggi di errore sono chiari e spiegano come risolvere il problema	Errori chiari
Efficienza	Ricerca e filtri	Il cliente può filtrare facilmente immobili per prezzo, zona, tipologia, ecc.	Filtraggio semplice
	Navigazione rapida	Le operazioni più comuni (cerca immobile, visualizza, carica immobile) sono veloci da eseguire	Operazioni comuni molto veloci
Accessibilità	Dimensione testo e pulsanti	Testo e pulsanti leggibili e facilmente cliccabili	Molto semplice

5.2.2 Esperimento con utenti reali

Questa sezione consente di valutare l'usabilità sia per utenti esperti sia per utenti inesperti, e per entrambi i ruoli dell'applicazione. L'esperimento ha coinvolto quattro partecipanti, suddivisi in due gruppi principali in base al ruolo previsto nell'applicazione. Gli utenti sono stati invitati a eseguire una serie di task rappresentativi dell'uso reale dell'app, osservando tempi, errori e difficoltà.

5.2.2.1 Task assegnati agli utenti

Di seguito sono riportati i task che ognuno dei quattro utenti

1) Registrazione all'applicazione tramite email.
2) Ricerca di un immobile filtrato ai propri criteri:
    a. Zona scelta dall'utente
    b. Budget a disposizione
3) Consultazione della posizione dell'immobile sulla mappa.
4) Tornare indietro alla lista degli immobili.
5) Effettuare il logout dall'app.

Il punto 2) e 3) per l'Agente sono sostituiti con il caricamento di un immobile

Per valutare l'usabilità sono state considerate e raccolte le seguenti metriche:

Metrica	Descrizione
Tempo di completamento	Tempo impiegato dall'utente per completare ciascun task
Numero di errori	Errori di navigazione, form, ricerca o selezione
Task completato	Task completato con successo / non completato
Facilità d'uso percepita	Valutazione soggettiva tramite survey post-task
Soddisfazione complessiva	Valutazione soggettiva dell'esperienza utente

5.2.2.2 Descrizione dei soggetti in esame

C1 – Cliente senza competenze tecniche
L'utente ha completato la fase di registrazione in circa 3 minuti, commettendo alcuni errori. Successivamente ha effettuato una ricerca di un immobile circa 1 minuto, riscontrando una difficoltà nell'utilizzo dei filtri di ricerca. La consultazione della posizione dell'immobile è avvenuta in poco tempo e senza problemi. Il logout è stato completato in pochi secondi, senza anomalie.

C2 – Cliente senza competenze tecniche
L'utente ha completato la registrazione in circa 3 minuti, con un errore. Anche in questo caso, durante la ricerca di un immobile sono emerse difficoltà nell'uso dei filtri. La consultazione della posizione è stata completata in circa 2 minuti senza problemi. Il logout è avvenuto correttamente in pochi secondi.

A1 – Agente immobiliare
L'agente ha completato la registrazione in circa 2 minuti, senza commettere errori. La fase di caricamento di un immobile ha richiesto circa 3 minuti e 30 secondi, con un lieve rallentamento riscontrato durante l'operazione. La consultazione del proprio immobile è stata effettuata in poco tempo e il logout è avvenuto rapidamente e senza problemi.

C3 – Cliente con competenze tecniche
L'utente ha completato la registrazione in circa 1 minuto, senza errori. La ricerca di un immobile è stata svolta in pochi secondi e la consultazione della posizione in 1 minuto, entrambe senza difficoltà. Anche il logout è stato eseguito rapidamente e senza problemi.

5.2.2.3 Risultati e discussioni

Il grafico (realizzato col tool Graph Maker di Canva) mostra l'andamento dell'apprendibilità del sistema durante l'esecuzione dei task principali.
Gli utenti inesperti presentano inizialmente un numero maggiore di errori e tempi più elevati, in particolare durante la fase di registrazione e ricerca. Tuttavia, la difficoltà diminuisce rapidamente nei task successivi, indicando una buona capacità di apprendimento del sistema.
Gli utenti esperti, invece, mostrano un andamento stabile con difficoltà minima sin dal primo utilizzo, a conferma di una buona usabilità complessiva dell'applicazione.

5.2.2.4 Survey somministrato

Infine, si riporta il sondaggio di gradimento presentato agli utenti, con una media di gradimento > 4

#	Domanda	1	2	3	4	5
1	La registrazione all'app è stata semplice e chiara	☐	☐	☐	☐	☐
2	Ho capito facilmente come cercare un immobile usando i filtri	☐	☐	☐	☐	☐
3	Le informazioni sugli immobili sono chiare e ben organizzate	☐	☐	☐	☐	☐
4	La consultazione della posizione dell'immobile sulla mappa è stata intuitiva	☐	☐	☐	☐	☐
5	La navigazione tra le varie schermate è fluida e comprensibile	☐	☐	☐	☐	☐
7	Nel complesso, l'applicazione è facile da usare	☐	☐	☐	☐	☐

6 Riferimenti bibliografici e fonti

I contenuti e la terminologia tecnica adoperata nei capitoli precedenti di questo documento sono stati recuperati dalle slide [1 - 40] del corso di “Ingegneria del Software” dell’A.A. 2024/25.

Altre fonti:

- Amazon Web Services. (2025). *AWS Cognito Developer Guide - Java Integration*. https://docs.aws.amazon.com/cognito/
- The Apache Software Foundation. (2025). *Maven POM Reference*. https://maven.apache.org/pom.html
- Oracle Corporation. (2025). *A Swing Architecture Overview*. https://docs.oracle.com/javase/tutorial/uiswing/
- JUnit Team. (2025). *JUnit 5 User Guide*. https://junit.org/junit5/docs/current/user-guide/