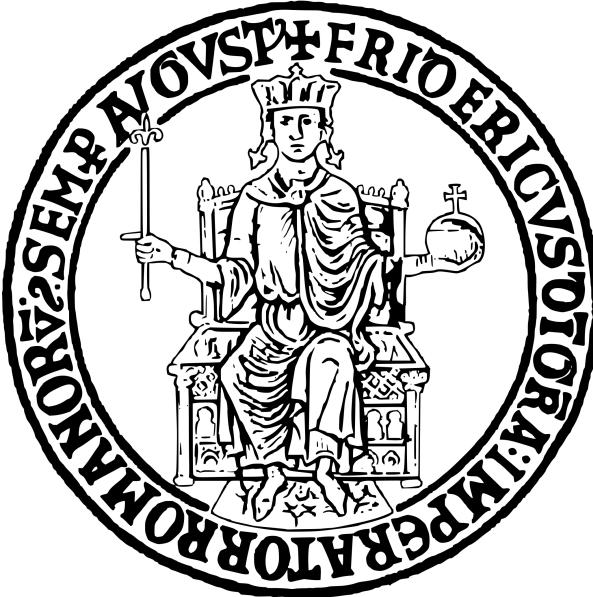


Università degli Studi di Napoli Federico II

Corso di Laurea in Informatica



DietiEstates25

Autori:

Nome Cognome 1 Matricola: 123456

Nome Cognome 2 Matricola: 654321

Nome Cognome 3 Matricola: 789012

Anno Accademico: 2024/2025

# Indice

|   |           |
|---|-----------|
| <b>Introduzione</b>   | <b>1</b>  |
| <b>1 Struttura della Documentazione</b>                                 | <b>1</b>  |
| <br>  |           |
| <b>Analisi dei requiti</b>  | <b>2</b>  |
| <b>2 Ruoli del sistema</b>  | <b>2</b>  |
| <b>3 Requisiti individuati</b>  | <b>2</b>  |
| 3.1 Requisiti non funzionali di DietiEstates25 . . . . .                | 2         |
| 3.2 Requisiti di dominio per DietiEstates25 . . . . .                   | 3         |
| <b>4 Modello dei casi d'uso</b>   | <b>5</b>  |
| 4.1 Tabella dei casi d'uso . . . . .                                    | 5         |
| 4.2 Descrizione dettagliata dei casi d'uso . . . . .                    | 6         |
| 4.3 Use Case Diagram . . . . .  | 10        |
| <b>5 Individuazione Target Utenti</b>                                   | <b>12</b> |
| 5.1 Personas: Antonio esposito . . . . .                                | 12        |
| 5.2 Personas: Daniela Formicola . . . . .                               | 13        |
| 5.3 Personas: Luca esposito . . . . .                                   | 14        |
| 5.4 Personas: Rosa bianchi . . . . .                                    | 15        |
| <b>6 Prototipazione visuale tramite Mock-up</b>                         | <b>16</b> |
| <b>7 Descrizione testuali strutturate</b>                               | <b>21</b> |
| 7.1 Tabella di Cockburn del caso d'uso n.1 e relativi Mock-up . . . . . | 21        |
| 7.2 Tabella di Cockburn del caso d'uso n.2 . . . . .                    | 23        |
| <b>8 Glossario</b>  | <b>24</b> |
| <br>  |           |
| <b>Documento di Design del Sistema</b>                                  | <b>25</b> |
| <br>  |           |
| <b>10 Archittettura del Software</b>                                    | <b>25</b> |
| 10.1 Client . . . . .   | 25        |
| 10.2 Server . . . . .   | 26        |
| 10.3 Struttura Schematizzata Back-End . . . . .                         | 27        |
| 10.4 Tecnologie Adottate . . . . .                                      | 28        |
| 10.5 JavaFX . . . . .   | 28        |
| 10.6 Jakarta Enterprise Edition . . . . .                               | 29        |

|  |           |
|--|-----------|
| 10.7 SceneBuilder . . . . .                  | 30        |
| 10.8 PostgreSQL . . . . .                    | 31        |
| 10.9 Docker . . . . .                        | 32        |
| 10.10 JavaScript . . . . .                   | 33        |
| 10.10.1 Perchè JavaScript? . . . . .         | 33        |
| 10.11 Altri servizi cloud usati . . . . .    | 34        |
| <b>11 Motivazione delle scelte di design</b> | <b>35</b> |
| 11.1 Intro . . . . .                         | 35        |
| 11.2 Colori del software . . . . .           | 35        |
| 11.3 Pagina di benvenuto . . . . .           | 36        |
| 11.4 Analisi del design . . . . .            | 36        |
| 11.5 Pagine di accesso . . . . .             | 37        |
| 11.6 Aree riservate . . . . .                | 38        |
| 11.7 Crea annuncio . . . . .                 | 38        |
| 11.8 Visualizza appuntamenti . . . . .       | 39        |
| 11.9 Modifica profilo . . . . .              | 40        |
| 11.10 Creazione account . . . . .            | 41        |
| 11.11 Visualizza notifiche . . . . .         | 42        |
| 11.12 Home di DietiEstates25 . . . . .       | 43        |
| 11.13 Ricerche passate . . . . .             | 43        |
| <b>12 Diagrammi delle classi di design</b>   | <b>45</b> |
| 12.1 Gestione delle notifiche . . . . .      | 45        |
| 12.2 Effettua prenotazione . . . . .         | 47        |
| 12.3 Registrazione . . . . .                 | 49        |
| 12.4 Cambio password . . . . .               | 51        |
| 12.5 Crea annuncio . . . . .                 | 53        |
| 12.6 Modifica profilo . . . . .              | 55        |
| 12.7 Autenticazione . . . . .                | 57        |
| 12.8 Ricerca . . . . .                       | 58        |
| 12.9 Creazione account . . . . .             | 60        |
| <b>13 Diagramma di sequenza di design</b>    | <b>62</b> |
| 13.1 Caso d'uso : Prenota visita . . . . .   | 62        |
| 13.2 Caso d'uso: Lascia recensione . . . . . | 64        |
| <b>Artefatti Software</b>                    | <b>65</b> |

|   |           |
|---|-----------|
| <b>14 DockerFile e file di build Automatica</b>         | <b>65</b> |
| 14.1 File di build Automatica . . . . .                 | 65        |
| 14.2 DockerFile . . . . .                               | 65        |
| 14.2.1 Struttura del Dockerfile . . . . .               | 65        |
| 14.2.2 Docker compose . . . . .                         | 66        |
| 14.2.3 6.1.3 Accesso Interattivo al Container . . . . . | 66        |
| 14.2.4 Gestione delle Password . . . . .                | 67        |
| <b>15 Strumento di versioning</b>                       | <b>67</b> |
| 15.1 Scelte adottate . . . . .                          | 67        |
| 15.2 Repository . . . . .                               | 68        |
| <b>16 Report di qualità del codice</b>                  | <b>69</b> |
| 16.1 SonarQube . . . . .                                | 69        |
| <b>Testing</b>  | <b>70</b> |
| 16.2 Unit Testing . . . . .                             | 70        |
| 16.2.1 lasciaRecensione . . . . .                       | 70        |
| 16.2.2 checkVisitaPerCliente . . . . .                  | 73        |
| <b>17 Usabilità sul campo</b>                           | <b>76</b> |
| <b>18 Valutazione degli esperti</b>                     | <b>80</b> |
| 18.1 Test con utenti reali . . . . .                    | 84        |
| <b>19 Risoluzione errori</b>                            | <b>85</b> |

# Introduzione

DietiEstates25 è un sistema software progettato con l'obiettivo di semplificare la gestione di servizi immobiliari. La piattaforma offre un'interfaccia intuitiva e comoda per semplificare la ricerca e l'interazione con annunci immobiliari, al fine di offrire all'utente un utilizzo rapido e piacevole.

## 1 Struttura della Documentazione

Il seguente documento è suddiviso in tre sezioni:

- Documento dei requisiti software: in questa sezione si analizzano a fondo le richieste fornite dal cliente elaborando con precisione tutti i requisiti richiesti per la realizzazione del software; Questa sezione rappresenta la prima parte fondamentale per la realizzazione del prodotto.
- Documento di Design del Sistema: in questa sezione è presente una descrizione e motivazione delle tecnologie adottate per la realizzazione del sistema.
- Testing e Valutazione dell'usabilità: in questa sezione è presente una descrizione e motivazione delle tecnologie adottate per la realizzazione del sistema.

# Analisi dei Requisiti

Per garantire la realizzazione di DietiEstates25, è fondamentale comprendere a fondo i requisiti del sistema. L'analisi dei requisiti verterà su:

- Identificazione degli attori principali del sistema
- Definizione dettagliata delle funzionalità chiave degli attori
- Definizione del target degli utenti
- Prototipazione dell'applicazione
- Studio dell'usabilità a priori

## 2 Ruoli del sistema

A seguito di un'attenta analisi delle esigenze del committente, sono stati individuati i seguenti ruoli all'interno del sistema:

- **Amministratore:** Ruolo di livello superiore nel sistema, responsabile della gestione generale dell'agenzia immobiliare. L'amministratore accede al sistema utilizzando credenziali predefinite e ha la possibilità di creare account per altri utenti, tra cui account di supporto.
- **Gestore dell'agenzia immobiliare:** Ruolo subordinato all'amministratore. L'account del gestore viene creato dall'amministratore e, a sua volta, il gestore ha la facoltà di creare account di supporto per altri utenti.
- **Agente immobiliare:** Rappresenta gli agenti immobiliari operanti nell'agenzia, con compiti legati alla gestione delle proprietà e alla comunicazione con i clienti.
- **Cliente:** Individui o entità che usufruiscono dei servizi offerti dagli agenti immobiliari, interessati alla ricerca e alla prenotazioni di visite.

## 3 Requisiti individuati

### 3.1 Requisiti non funzionali di DietiEstates25

- Il sistema deve essere sviluppato utilizzando un linguaggio Object-Oriented.
- Il sistema deve offrire un'interfaccia intuitiva e facile da utilizzare, con una chiara visualizzazione delle informazioni relative agli immobili.
- Il sistema deve garantire una ricerca rapida ed efficiente. In particolare, il tempo di risposta della ricerca deve essere inferiore a 2 secondi nel 99% delle richieste, in modo da assicurare un requisito testabile per il futuro.

- Il sistema deve conservare le credenziali degli utenti in modo sicuro.
- Il sistema deve essere semplice da utilizzare. In particolare:
  - I clienti che utilizzano l'applicazione devono poter usufruire di tutte le funzionalità principali (login, prenotazione visita, visualizzazione notifiche, ecc.) in modo immediato, senza necessità di formazione.
  - Gli agenti immobiliari e i gestori devono poter acquisire familiarità con tutte le funzionalità del sistema entro un massimo di 30 minuti di formazione. Dopo tale periodo, il numero di errori commessi non dovrebbe superare 1 all'ora durante l'utilizzo del sistema.
  - Nessun utente dovrebbe avere la necessità di richiedere supporto per l'uso ordinario del sistema.
- Il sistema deve garantire un tempo di disponibilità del 99% su base mensile.

### **3.2 Requisiti di dominio per DietiEstates25**

- Il sistema, in fase di creazione dell'annuncio, deve verificare che l'utente abbia inserito la classe energetica in conformità con il Decreto Legislativo 192/2005.
- Solo gli agenti immobiliari possono caricare annunci di immobili; tutti gli altri utenti non hanno questa possibilità.
- Gli amministratori, i gestori e gli agenti non possono prenotare visite agli immobili.
- Gli amministratori e i gestori non possono lasciare recensioni agli agenti immobiliari.
- Gli agenti immobiliari non possono lasciare recensioni ad altri agenti né a sé stessi.
- Gli immobili devono essere classificati in categorie come appartamenti, ville, uffici e locali commerciali.
- Il sistema deve garantire che ogni annuncio immobiliare contenga informazioni obbligatorie secondo la normativa vigente, tra cui classe energetica, metratura, numero di stanze e prezzo.



## 4 Modello dei casi d'uso

### 4.1 Tabella dei casi d'uso

| Attore                      | Casi d'uso   |
|-----------------------------|--|
| Utente                      | <ul style="list-style-type: none"><li>• Accede al sistema</li><li>• Ricerca immobili</li><li>• Modifica profilo personale</li><li>• Visualizza notifiche</li><li>• Accede tramite credenziali di terze parti</li></ul> |
| Cliente                     | <ul style="list-style-type: none"><li>• Lascia recensione</li><li>• Prenota visita</li></ul>   |
| Amministratore              | <ul style="list-style-type: none"><li>• Crea account supporto</li></ul>  |
| Gestore azienda immobiliare | <ul style="list-style-type: none"><li>• Crea account per gli agenti immobiliari</li></ul>  |
| Agente immobiliare          | <ul style="list-style-type: none"><li>• Carica immobile</li><li>• Accetta visita</li><li>• Rifiuta visita</li><li>• Visualizza appuntamenti</li><li>• Aggiunge foto profilo</li></ul>                                  |
| Utente non registrato       | <ul style="list-style-type: none"><li>• Registra al sistema</li><li>• Registra al sistema tramite credenziali di terze parti</li></ul>   |

## 4.2 Descrizione dettagliata dei casi d'uso

### Utente

- **Accede al sistema:** L'utente inserisce email e password e accede al sistema.
- **Accede tramite google:** L'utente può cliccare "*Continua con google*" e accedere con le sue credenziali
- **Ricerca immobili tramite parametri:** L'utente può cercare immobili filtrando per criteri come posizione, prezzo, numero di stanze e tipologia di immobile. Il sistema restituisce un elenco di risultati corrispondenti con dettagli degli immobili cercati e immagini corrispondenti.
- **Modifica dati:** L'utente andando nella sezione "Modifica profilo" può visualizzare i propri dati e modificarli. Può modificare informazioni relativi alla sua email, password ma anche nome, cognome e numero di telefono.
- **Visualizza notifiche:** L'utente andando nella sezione "Notifiche" può visualizzare le sue notifiche e optionalmente può anche decidere di attivare o disattivare singole categorie di notifiche. In base a che ruolo ricoprirà l'utente si andranno a visualizzare diverse categorie di notifiche.
- **Accede tramite google:** L'utente può accedere al sistema tramite google. Una volta effettuato l'accesso se questo ha avuto successo, l'utente accede al sistema.

### Cliente

- **Prenota visita:** Il cliente può cliccare sull'immobile interessato, una volta cliccato gli vengono visualizzate tutte le informazioni sull'immobile e può cliccare su "Prenota visita". Seleziona una data, seleziona una fascia oraria e clicca prenota
- **Lascia recensione:** Il cliente può cliccare sull'immobile interessato, una volta cliccato gli vengono visualizzate tutte le informazioni sull'immobile, insieme alle informazioni sull'immobile vi è anche un'area dedicata all'agente che ha caricato l'immobile. L'utente clicca "Lascia recensione", seleziona il numero di stelle che vuole dare all'agente e clicca "Invia".

### Amministratore

- **Creazione di account di supporto:** L'amministratore accede al sistema e seleziona l'opzione "Crea gestori". Una volta avviata la procedura, gli verrà presentato un modulo di registrazione, nel quale dovrà inserire i dati anagrafici del gestore da creare. Le informazioni richieste sono le seguenti:
  - Nome
  - Cognome
  - Codice Fiscale
  - Numero di telefono

- Email
- Password

La decisione di far inserire i dati anagrafici all'amministratore nella fase di creazione dell'account è motivata dalla necessità di garantire un'organizzazione e una gestione efficace del sistema. In qualità di responsabile della gestione del personale dell'agenzia immobiliare, l'amministratore è l'unico autorizzato alla creazione degli account per i gestori. Essendo una figura centrale nell'organizzazione, deve avere accesso completo alle informazioni anagrafiche dei dipendenti sin dall'inizio del processo. Questo approccio garantisce una gestione centralizzata e sicura degli utenti, riduce il rischio di errori nella registrazione e ottimizza i flussi operativi, permettendo l'attivazione immediata degli account al primo accesso dei gestori.

### Gestore dell'agenzia immobiliare

- **Creazione di account per gli agenti immobiliari:** Il gestore accede al sistema e seleziona l'opzione "Crea agenti". Una volta avviata la procedura, gli verrà presentato un modulo di registrazione, nel quale dovrà inserire i dati anagrafici dell'agente da creare. Le informazioni richieste sono le seguenti:

- Nome
- Cognome
- Codice Fiscale
- Numero di telefono
- Email
- Password

Abbiamo adottato lo stesso principio seguito nella fase di creazione degli account per i gestori. Poiché il gestore è responsabile della gestione diretta degli agenti immobiliari, è essenziale che disponga di tutte le informazioni anagrafiche necessarie per garantire una gestione strutturata ed efficace del personale. Questo metodo assicura un controllo centralizzato, riduce la possibilità di errori e favorisce un accesso immediato e sicuro al sistema per gli agenti.

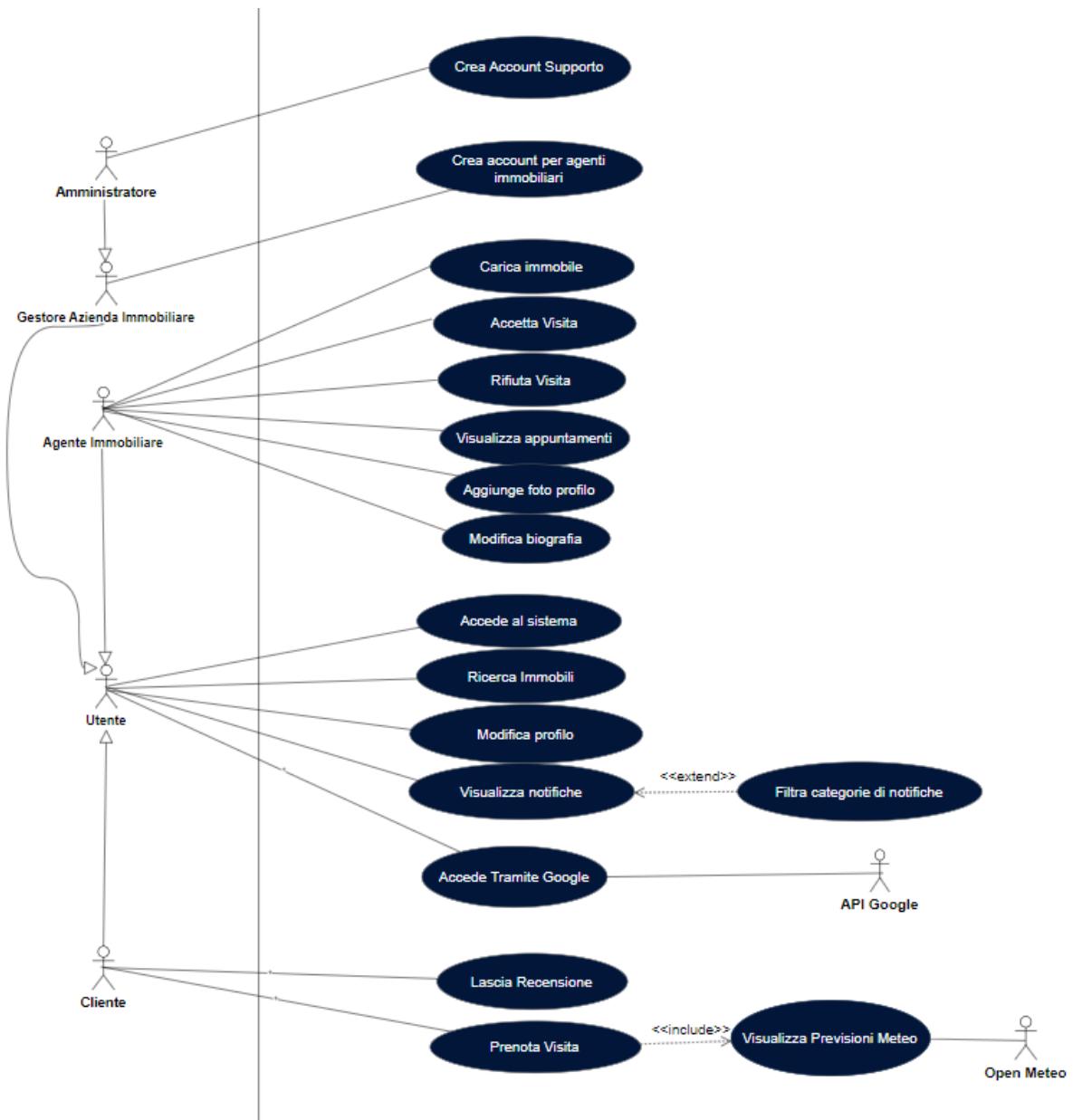
## Agente immobiliare

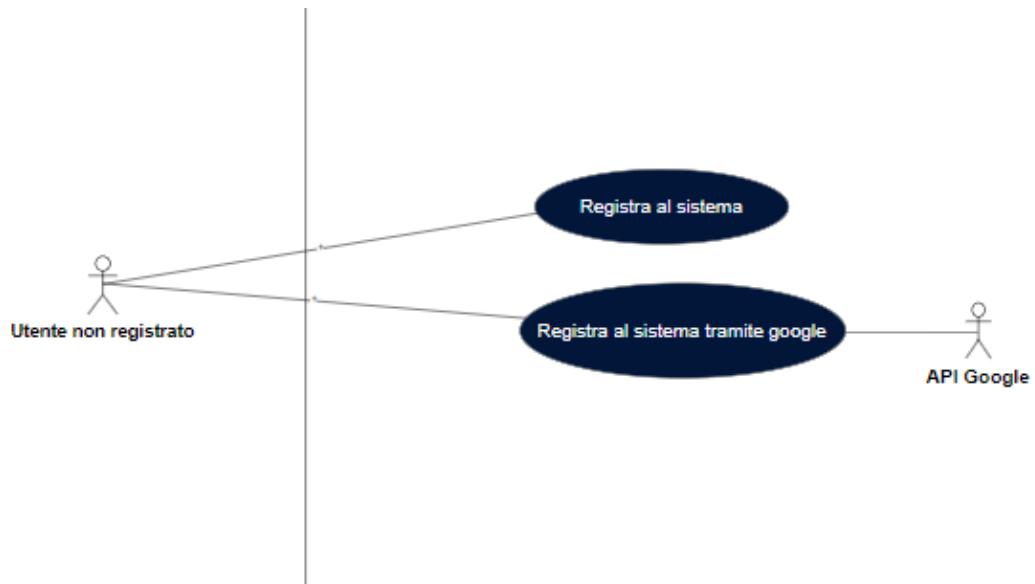
- **Caricamento di un immobile:** L'agente accede al sistema e seleziona l'opzione "*Carica immobile*". Durante questa operazione, è tenuto a inserire una serie di informazioni dettagliate relative all'immobile, tra cui:
  - **Tipologia:** specifica se si tratta di un appartamento, una villa, un ufficio, ecc.
  - **Tipo di vendita:** indica se l'immobile è in vendita, in affitto.
  - **Posizione:** comprende l'indirizzo e la localizzazione geografica dell'immobile.
  - **Dimensioni:** superficie totale espressa in metri quadrati.
  - **Piano:** numero del piano in cui si trova l'immobile.
  - **Numero di locali:** numero totale di stanze presenti nell'immobile.
  - **Classe energetica:** certificazione dell'efficienza energetica dell'immobile.
  - **Prezzo:** valore richiesto per la vendita o l'affitto.
  - **Spese condominiali:** eventuali costi mensili per la gestione del condominio.
  - **Presenza di ascensore:** indica se l'immobile è dotato di ascensore.
  - **Fotografie:** immagini rappresentative dell'immobile per una migliore presentazione.
  - **Descrizione:** testo descrittivo che fornisce dettagli aggiuntivi sulle caratteristiche e i punti di forza dell'immobile.
- **Accettazione di una visita:** l'agente accede al sistema e seleziona la sezione "*Notifiche*", dove può visualizzare le richieste di visita ricevute. Dopo aver esaminato le notifiche, può confermare un appuntamento selezionando l'opzione "*Accetta visita*".
- **Rifiuto di una visita:** l'agente accede al sistema e seleziona la sezione "*Notifiche*", dove può visualizzare le richieste di visita ricevute. Dopo aver esaminato le notifiche, può confermare un appuntamento selezionando l'opzione "*Rifiuta visita*".
- **Visualizzazione degli appuntamenti:** l'agente accede al sistema e seleziona l'opzione "*Visualizza appuntamenti*". In questa sezione, può consultare l'elenco delle visite accettate e organizzare al meglio la propria agenda.
- **Modifica della foto profilo:** l'agente accede al sistema e seleziona l'opzione "*Modifica profilo*". Viene reindirizzato a una pagina in cui è visibile l'immagine attualmente impostata. Da questa sezione, può selezionare l'opzione "*Modifica foto*" e caricare una nuova immagine di profilo.
- **Modifica biografia :** L'agente accede al sistema e seleziona l'opzione "*Modifica profilo*". Viene reindirizzato a una pagina in cui è visibile la propria biografia attuale.Da questa sezione può selezionare l'opzione "*Modifica biografia*",l'agente carica la sua biografia il sistema fa una serie di controlli(come la lunghezza) e la biografia viene aggiornata.

## **Utente non registrato**

- **Registrazione:** Un nuovo utente può creare un account fornendo nome,cognome,email e password. I dati inseriti sono soggetti a verifica da parte del sistema (ad esempio: la email deve essere in un formato valido o la password deve essere abbastanza sicura). Una volta effettuata la registrazione, se questa ha avuto successo, l'utente accede al sistema
- **Registrazione tramite google:**Un nuovo utente può cliccare "*Continua con google*" e accedere con le sue credenziali di terze parti.

### 4.3 Use Case Diagram





## 5 Individuazione Target Utenti

Durante la prima fase di sviluppo dell'intero software è cruciale il segmento di individuazione e caratterizzazione del Target degli utenti che utilizzeranno il software, in modo tale da soddisfare tutte le richieste e necessità presenti.

**Personas:** affinché si abbia un'idea concreta dei possibili utenti del nostro software e delle necessità che potrebbero avere nell'utilizzo di esso, sono stati creati dei profili di futuri utenti del nostro prodotto. Per la creazione di questi profili abbiamo estratto da diverse piattaforme, quali presentano obiettivi e utilizzo simile al quello del nostro software, informazioni di base degli utenti presenti.

### 5.1 Personas: Antonio esposito



Il primo profilo identificato è quello di un lavoratore di 40 anni, alla ricerca di una casa adatta alle nuove esigenze della sua famiglia. Durante l'esperienza precedente con altre piattaforme, il principale problema riscontrato da Antonio Esposito riguarda l'interfaccia utente, che non risultava sufficientemente intuitiva e facile da usare. Pertanto, è necessario sviluppare un'interfaccia che sia semplice e chiara, in grado di consentire agli utenti di effettuare ricerche mirate e di ottenere esclusivamente i risultati più pertinenti alle loro necessità, evitando risultati superflui.

## 5.2 Personas: Daniela Formicola



### Daniela Formicola

**About**

23  
Napoli  
Studentessa  
Università Federico II Di Napoli  
Single  
Meticolosa  
Assertiva  
Energetica  
Ansiosa  
Insicura  
Perseverante

**Descrizione**

Daniela Formicola è una studentessa di ingegneria meccanica di 23 anni. Vive con i genitori e lavora part-time presso una palestra. Ha recentemente conseguito la laurea triennale ed è ora pronta a proseguire gli studi con una laurea magistrale. Venendo a conoscenza che la sede della nuova università si trova fuori città, desidera utilizzare un sistema per cercare immobili nelle vicinanze. Non avendo esperienza nel settore immobiliare, Daniela si sente insicura e preoccupata riguardo alla possibilità di trovare una soluzione abitativa adatta alle sue esigenze, il che le causa ansia e stress nella ricerca del nuovo immobile.

**Personalità**

| Introvert | Extrovert   |
|-----------|-------------|
| Thinking  | Feeling     |
| Judging   | Prospecting |
| Assertive | Turbulent   |
| Intuitive | Observant   |

**Obiettivi**

- Trovare un immobile vicino la sua nuova Università

**Criticità**

- La principale difficoltà per Daniela potrebbe derivare dal fatto che si affaccia per la prima volta al mondo immobiliare. Essendo inesperta, potrebbe incontrare difficoltà nella ricerca dell'immobile e sentirsi disorientata di fronte ai prezzi e alle varie complessità per comprare un immobile, il che contribuirebbe a renderla ansiosa e restia a utilizzare il sistema. È quindi prioritario garantire a Daniela un'esperienza utente rassicurante e intuitiva. Il sistema dovrebbe fornire feedback chiari e immediati per ridurre la sua insicurezza e presentare una navigazione semplificata, limitando le informazioni superflue, così da alleviare la sua ansia nell'interazione con la piattaforma.

**Skills**

| Internet            | Social Networks     | Online Shopping     |
|---------------------|---------------------|---------------------|
| ● ● ● ● ● ● ● ● ● ● | ● ● ● ● ● ● ● ● ● ● | ● ● ● ● ● ● ● ● ● ● |

Il secondo profilo identificato è quello di Daniela Formicola, una neolaureata triennale di 23 anni, attualmente alla ricerca di una sistemazione nelle vicinanze della sua nuova sede universitaria. Il suo obiettivo è affrontare questo processo in modo semplice e privo di stress.

Dall'analisi della sua esperienza, è emerso come principale criticità la **scarsa familiarità con l'utilizzo delle piattaforme di ricerca immobiliare, unita a una conoscenza limitata del settore**, trattandosi della sua prima esperienza in questo ambito.

Per questo motivo, risulta essenziale progettare un'interfaccia intuitiva, chiara ed essenziale, priva di informazioni superflue che potrebbero generare confusione. Inoltre, è fondamentale che la piattaforma sia in grado di presentare le informazioni in modo semplice e accessibile, considerando che una parte degli utenti potrebbe trovarsi per la prima volta ad affrontare un processo di ricerca immobiliare.

### 5.3 Personas: Luca esposito



## Luca Esposito

### About

45  
Napoli  
Agente immobiliare  
Diplomato  
Sposato

Meticoloso Assertivo Pragmatico  
Efficiente Carismatico Perseverante

### Descrizione

Luca è un agente immobiliare presso un'agenzia con una buona reputazione locale. Usa spesso portali immobiliari per pubblicizzare le proprietà che rappresenta, ma trova i tool attuali limitati e non sempre orientati all'ottimizzazione della vendita.

### Obiettivi

- Avere strumenti per pubblicare annunci di alta qualità con la possibilità di interagire facilmente con potenziali acquirenti o affittuari.
- Risparmiare tempo tramite strumenti di gestione degli appuntamenti dei clienti.

### Criticità

- Attualmente, deve caricare manualmente annunci su diversi portali per raggiungere un pubblico ampio. Mancano strumenti di feedback degli utenti per migliorare le strategie di vendita. Perde parecchio tempo per fissare gli appuntamenti per la visita dei vari immobili.

### Personalità

| Introvert | Extrovert   |
|-----------|-------------|
| Thinking  | Feeling     |
| Judging   | Prospecting |
| Assertive | Turbulent   |
| Intuitive | Observant   |

### Skills

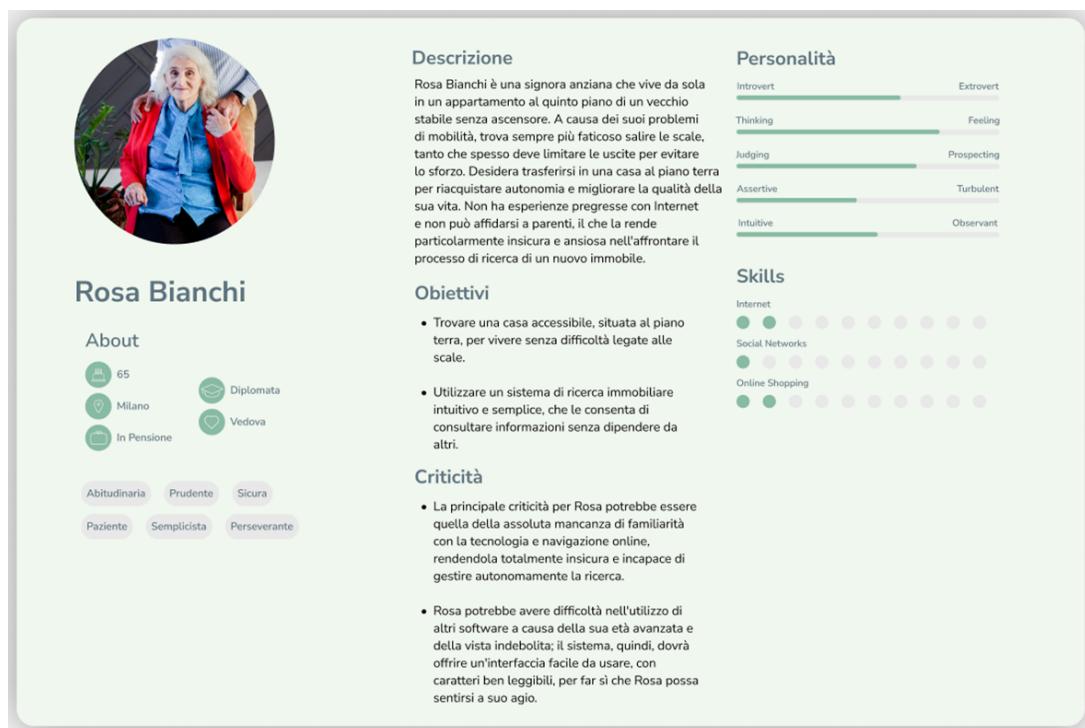
| Internet        |
|-----------------|
| Social Networks |
| Online Shopping |

Il terzo profilo identificato è quello di Luca Esposito, un agente immobiliare di 45 anni, il cui interesse principale è rivolto alla velocità, all'ottimizzazione e alla qualità nel processo di gestione degli immobili.

Dall'analisi della sua esperienza, è emerso che Luca riscontra **difficoltà nell'utilizzo di altre piattaforme**, in particolare nella procedura di caricamento degli immobili, che spesso risulta complessa e poco intuitiva. Inoltre, le interfacce utente delle soluzioni attualmente disponibili presentano un eccesso di informazioni, rendendo l'interazione più onerosa.

Alla luce di queste considerazioni, il processo di caricamento degli immobili deve essere progettato in modo da risultare **semplice e immediato**, evitando la necessità per gli agenti immobiliari di consultare documentazione aggiuntiva o di affrontare complessità operative superflue.

## 5.4 Personas: Rosa bianchi



Il quarto profilo identificato è quello di Rosa Bianchi, una donna di 65 anni che vive da sola ed è alla ricerca di un'abitazione accessibile e confortevole, in grado di rispondere alle sue specifiche esigenze.

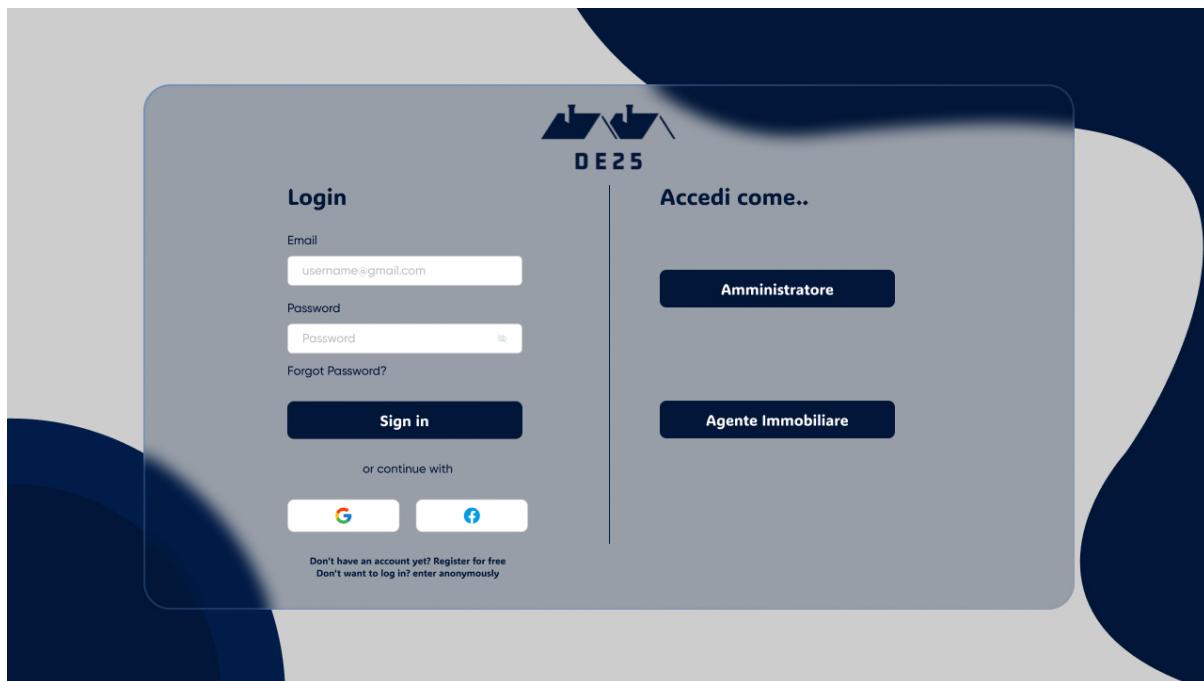
Dall'analisi della sua esperienza, è emersa una limitata **familiarità con l'uso delle piattaforme digitali**, evidenziando la necessità di sviluppare un'interfaccia intuitiva, chiara e facilmente leggibile. L'obiettivo è garantire una navigazione semplice e accessibile, anche per utenti con ridotte competenze informatiche o **difficoltà visive legate all'età**, facilitando il processo di ricerca dell'abitazione ideale senza ostacoli.

Alla luce di queste considerazioni, il sistema dovrà essere progettato in modo da presentare le informazioni in modo chiaro e leggibile, tenendo conto delle esigenze di utenti che potrebbero riscontrare difficoltà nella lettura e nell'interazione con le interfacce digitali.

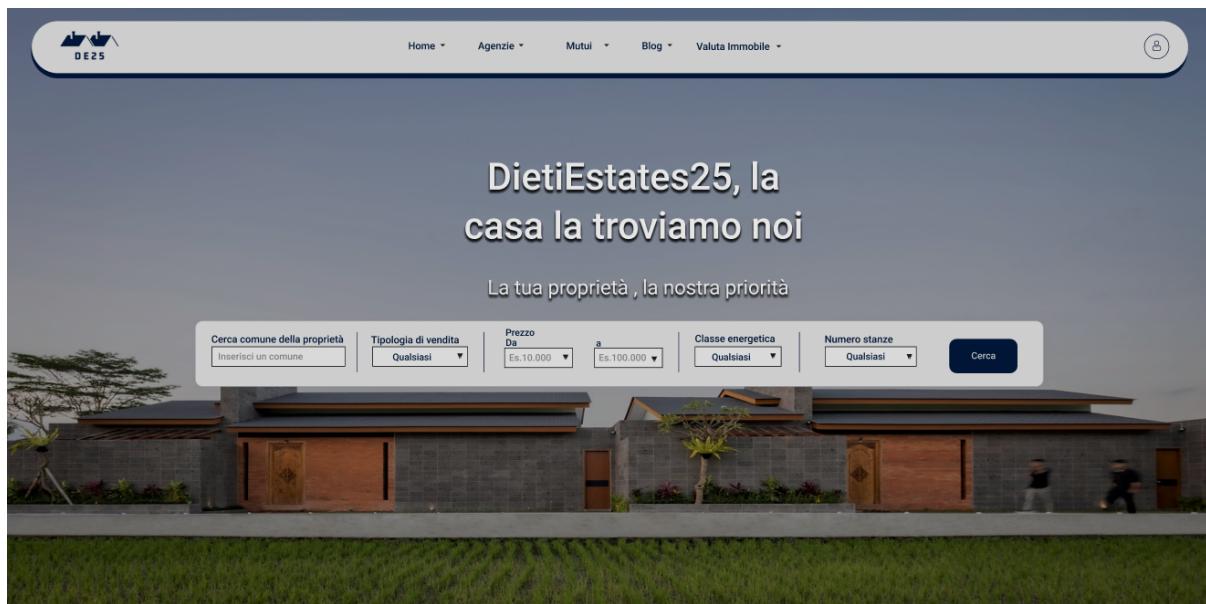
## 6 Prototipazione visuale tramite Mock-up

### Prototipazione Visuale attraverso Mock-Up

In questa sezione, verranno presentati i Mock-up realizzati per delineare il flusso delle principali funzionalità di DietiEstates25. Questi prototipi offrono una rappresentazione visiva delle schermate dell'applicazione e costituiscono uno strumento essenziale per raccogliere feedback preliminari sugli aspetti estetici e funzionali dell'interfaccia. Grazie a questi modelli, è possibile analizzare e ottimizzare l'esperienza utente prima dello sviluppo definitivo. **Si tende a specificare che questi mockup non sono la versione definitiva del sistema.**



Mock-up M1 Pagina di benvenuto



**Mock-up M2 Home del sistema**

The image shows a screenshot of the DietiEstates25 search results page for properties in Naples. The header is identical to the homepage. The search bar shows 'Napoli - Comune'. Below the search bar are filters for 'Tutti i Filtri' (All filters), 'Appartamenti' (Apartments), 'Prezzo' (Price), 'Compra' (Purchase), and 'Superficie' (Surface). The main content area displays two property listings. The first listing is for an apartment at Via Francesco Caracciolo, Mergellina, Naples, priced at €1,300,000. It includes a photo of the exterior, interior photos, and details about the property: 2 beds, 6 rooms, 230 m², 1 floor. The second listing is for an apartment at Via Firenze, Naples, priced at €375,000. It also includes a photo of the exterior and interior photos. To the right of the listings is a map of Naples with various neighborhoods labeled: Giugliano in Campania, Melito di Napoli, Scampia, Pianura, Fuorigrotta, Bagnoli, Posillipo, Ponticelli, Afragola, Frattamaggiore, and San Giorgio a Cremano. There are zoom controls (+/-) and a 'Espandi' (Expand) button on the map.

**Mock-up M3 Risultati ricerca**

**Contatta l'inserzionista**

Vorrei sapere di più su questo immobile, la contatto per maggiori informazioni

Invia il tuo messaggio

Prenota visita

Visualizza profilo

Lascia recensione

**Tecnocasa GROUP**

**Appartamento Via Francesco Caracciolo, Mergellina - Napoli** € 1.300.000

**Descrizione**

Situato in Via Francesco Caracciolo, nel cuore di Mergellina, questo esclusivo appartamento al primo

Mock-up M4 Visualizza immobile

Lascia la tua recensione ad Elia Rossetti

Digita qui...

Valuta agente

1 2 3 4 5

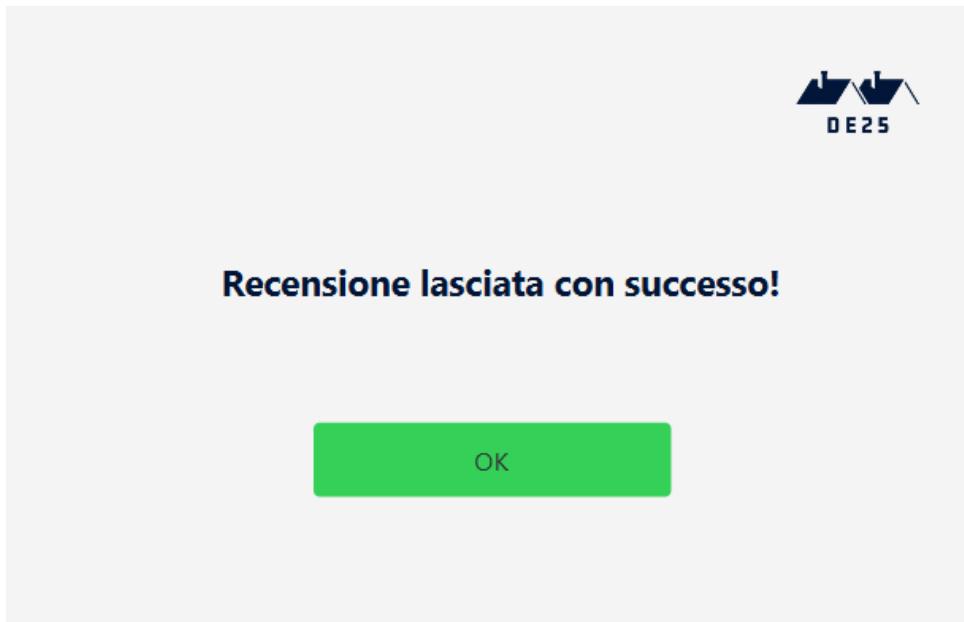
**Appartamento Via Francesco Caracciolo, Mergellina - Napoli** € 1.300.000

**Descrizione**

Situato in Via Francesco Caracciolo, nel cuore di Mergellina, questo esclusivo appartamento al primo

Mock-up M5 Lascia recensione

Osservazione: Si noti che il pulsante "Invia" è disabilitato fin quando l'utente non digita un testo e inserisce una valutazione per l'agente.



Mock-up M6 Successo Lascia recensione

A screenshot of a web application interface. At the top, there's a navigation bar with links for Home, Agenzie, Mutui, Blog, and Valuta Immobile. On the right side of the header are icons for user profile, message, and search. Below the header, there's a sidebar on the left showing a thumbnail of an apartment interior and some descriptive text. The main content area is a weather forecast card for Saturday, November 2, 2024, in Naples, Italy, with a 75% chance of rain. It includes a "Scegli un giorno" section with buttons for Saturday, Sunday, Monday, and Tuesday, and a "Scegli una fascia oraria" section with time slots from 9:00 to 19:00. On the right side of the main content area, there's a vertical sidebar with buttons for "Lascia recensione" and other user actions. A green "Proseguì" button is located at the bottom right of the main content area.

Mock-up M7 Prenota visita

Osservazione: Si noti che il pulsante "Proseguì" è disabilitato fin quando l'utente non seleziona un giorno per visita e una fascia oraria.



**Prenotazione effettuata con successo!**

OK

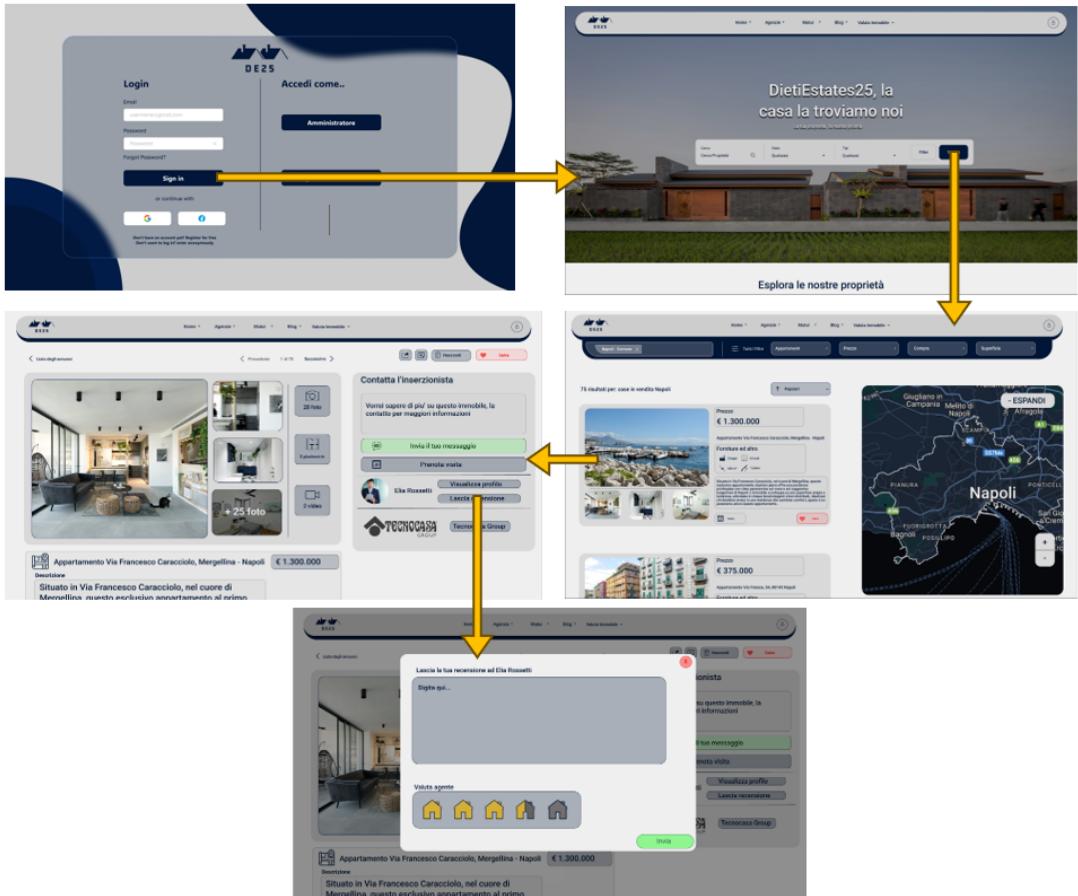
Mock-up M8 Successo Prenota visita

## 7 Descrizione testuali strutturate

In questa sezione verranno analizzati in dettaglio alcuni dei principali casi d'uso identificati per il sistema. L'analisi sarà condotta attraverso descrizioni testuali strutturate, utilizzando un formalismo tabellare basato sul modello di Cockburn, rappresentando in modo chiaro e sistematico l'interazione tra l'utente e il software, le condizioni di avvio e di successo, gli attori coinvolti e le eventuali eccezioni.

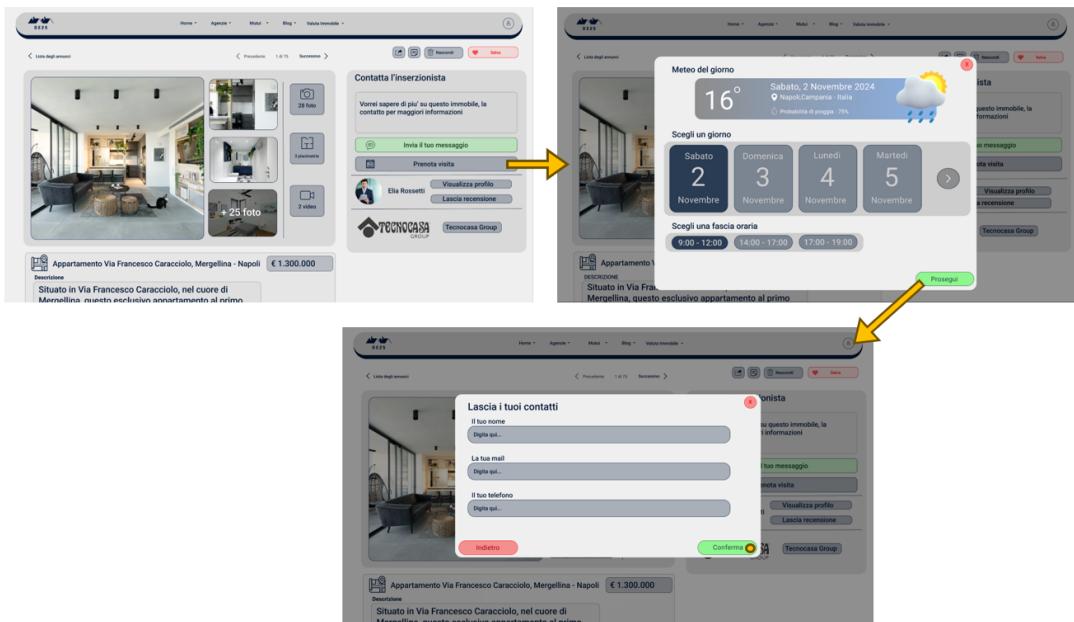
### 7.1 Tabella di Cockburn del caso d'uso n.1 e relativi Mock-up

| USE CASE #1 <i>Lascia Recensione Ad Agente</i>                                      |  |   |  |
|---|--|---|--|
| <b>Goal in Context</b>  | Il cliente potrà dare una valutazione sul servizio offerto dall'agente immobiliare   |   |  |
| <b>Preconditions</b>  | L'utente deve aver effettuato l'accesso o deve essere registrato; L'utente deve aver effettuato una ricerca ed aver cliccato su un immobile. |   |  |
| <b>Success End Condition</b>  | Il Sistema tiene conto della recensione; recensione salvata; recensione appare sulla lista delle recensioni dell'agente                      |   |  |
| <b>Failed End Condition</b>   | Il Sistema non tiene conto della recensione  |   |  |
| <b>Primary Actor</b>  | Cliente  |   |  |
| <b>Trigger</b>  | Cliente clicca lascia recensione in M4 "Visualizza immobile"   |   |  |
| <b>Main Scenario</b>  | <b>Step n.</b>   | <b>Client</b>   | <b>System</b>  |
|   | 1  | Clicca lascia recensione in M4<br>"Visualizza immobile" |  |
|   | 2  |   | Mostra M5 "Lascia recensione"  |
|   | 3  | Scrive Recensione; Inserisce<br>Numero Di Stelle        |  |
|   | 4  | Clicca Invia Valutazione                                |  |
|   | 5  |   | Mostra M6 "Successo lascia<br>recensione"  |
|   | 6  | Clicca "OK"   |  |
|   | 7  |   | Mostra M4 "Visualizza<br>immobile"   |
| <b>Extension Point A<br/>(Cliente clicca 'X' in<br/>M5 "Lascia<br/>recensione")</b> | <b>Step n.</b>   | <b>Client</b>   | <b>System</b>  |
|   | 1b   | Clicca 'X'  |  |
|   | 2b   |   | Mostra M4 "Visualizza<br>immobile" e termina UC  |
| <b>Extension Point B<br/>(Cliente Inserisce Un<br/>Testo Non Valido)</b>            | <b>Step n.</b>   | <b>Client</b>   | <b>System</b>  |
|   | 1c   | Inserisce un testo non valido                           |  |
|   | 2c   | Selezione numero di stelle                              |  |
|   | 3c   | Clicca invia valutazione                                |  |
|   | 4c   |   | Mostra Un Messaggio Di Errore<br>a ridosso dei form e ritorno allo<br>step 3 nel Main Scenario |



## 7.2 Tabella di Cockburn del caso d'uso n.2

| USE CASE #2  | Prenota Visita   |   |
|--|--|---|
| Goal in Context  | Il cliente vuole prenotare una visita di un immobile.  |   |
| Preconditions  | L'utente deve aver effettuato l'accesso o deve essere registrato; Il cliente deve aver cercato un immobile; Il cliente deve cliccare su un immobile. |   |
| Success End Condition  | Il sistema dovrà tenere traccia della visita; Il sistema dovrà registrarla negli appuntamenti dell'agente; il sistema dovrà notificare l'agente.     |   |
| Failed End Condition   | La prenotazione non viene effettuata; il sistema non tiene traccia della prenotazione; non arriva nessuna notifica all'agente.                       |   |
| Primary Actor  | Cliente  |   |
| Trigger  | Il cliente clicca prenota una visita in M4 "Visualizza immobile"   |   |
| Main Scenario  | Step n.  | Cliente   |
|  | 1  | Clicca "Prenota visita" in M4 "Visualizza Immobile" |
|  | 2  | Mostra M7 "Prenota Visita"                          |
|  | 3  | Selezione il giorno                                 |
|  | 4  | Selezione la fascia oraria                          |
|  | 5  | Clicca "proseguì"                                   |
|  | 6  | Mostra M8 "Messaggio di conferma"                   |
|  | 7  | Clicca 'Ok'   |
|  | 8  | Mostra M4 "Visualizza Immobile" e termina UC        |
| Extension Point A<br>(Cliente "X" In M7<br>"Prenota visita") | Step n.  | System  |
|  | 1a   | Clicca "X" in M7 "Prenota visita"                   |
|  | 2a   | Ritorna a M4 "Visualizza Immobile" e termina UC     |



## 8 Glossario

- **UML (Unified Modeling Language)** : Un linguaggio di modellazione grafica standard utilizzato per visualizzare, specificare, costruire e documentare gli artefatti di un sistema software. Include vari tipi di diagrammi, come diagrammi delle classi, dei casi d'uso e delle sequenze.
- **Mockup** : Rappresentazione statica dell'interfaccia utente di un'applicazione, utilizzata per mostrare il layout visivo e simulare l'aspetto dell'interfaccia prima della realizzazione finale.
- **Casi d'uso** : Tecnica di analisi funzionale che descrive, in termini di attori e scenari, come il sistema interagisce con gli utenti per soddisfare determinati obiettivi. Ogni caso d'uso documenta le interazioni tra l'attore e il sistema.
- **Tabelle di cockburn** : Modello di rappresentazione dei casi d'uso proposto da Alistair Cockburn. Ogni tabella contiene informazioni come titolo, attore principale, precondizioni, flusso principale di eventi, estensioni e risultati attesi, rendendo i casi d'uso più strutturati e comprensibili.
- **Attore** : Entità esterna al sistema che interagisce con esso. Può essere un utente umano o un altro sistema che richiede servizi al software.
- **Personas** : sono rappresentazioni fittizie di utenti ideali, basate su dati reali, che aiutano a progettare prodotti, servizi o interfacce in modo più mirato.
- **Use Case Diagram** : Diagramma UML che rappresenta le interazioni tra gli attori e il sistema, mostrando le funzionalità principali (casi d'uso) offerte dal sistema
- **API(Application Programming Interface)** : Interfaccia che permette a diverse applicazioni di comunicare tra loro.
- **Sequence Diagram** : Diagramma UML che mostra come gli oggetti interagiscono in una sequenza temporale.
- **Front-end**: Parte di un'applicazione software che interagisce direttamente con l'utente.
- **Back-end** : Parte di un'applicazione software che gestisce la logica, i database e le operazioni di background.

# Documento di Design del Sistema

In questa sezione vengono illustrate le scelte architetturali e i design pattern adottati nel software. Un design pattern è una soluzione generale e collaudata per risolvere problemi ricorrenti che si incontrano durante la progettazione di software. Questi pattern possono migliorare la comprensione del codice, la manutenzione e la scalabilità di un'applicazione.

## 10 Archittettura del Software

Il software sviluppato adotta un'architettura client-server, in cui il front-end è responsabile della presentazione dei dati e interagisce con il back-end tramite le API esposte. Il back-end fornisce i servizi e la logica di business, rendendo disponibili le API accessibili tramite rete, mantenendo così un'indipendenza strutturale rispetto alla parte front-end. La parte front-end è stata sviluppata utilizzando **JavaFX** e il server utilizzando **AWS**.

Sulla base delle specifiche appena esposte il sistema è organizzato in due sottosistemi principali:

- **Client:** Si occupa dell'interazione con l'utente, inviando richieste al *Server* ed elaborandone le risposte.
- **Server:** Gestisce le richieste pervenute dal *Client*, fa rispettare i vincoli esposti dalla business logic, interagisce con la base di dati relazionale per la persistenza dei dati e invia le risposte al *Client*.

### 10.1 Client

Il front-end è stato sviluppato utilizzando JavaFX. L'archittetura del front-end segue un approccio ben strutturato e diviso in strati.

- **FXML (View):** Gli FXML definiscono l'interfaccia utente (UI) in modo dichiarativo, separando la logica di presentazione dalla logica di business. Ogni file FXML è collegato a un controller che gestisce gli eventi e le interazioni dell'utente.
- **Controller:** Questo livello gestisce le interazioni con l'utente, come i click sui pulsanti o l'inserimento dei dati. Non contiene logica di business, ma delega le operazioni a un ulteriore livello, il Facade.
- **Facade:** Questo strato funge da punto di accesso semplificato tra il front-end e il back-end. Il Facade incapsula la complessità delle chiamate al back-end e fornisce un'interfaccia astratta ai controller, i quali non devono preoccuparsi dei dettagli di comunicazione con il back-end. A sua volta, il Facade interagisce con il livello Service.
- **Service:** Questo livello si occupa della comunicazione con il back-end, effettuando le chiamate alle API esposte dal server e restituendo i dati elaborati al Facade.

## Perché questa architettura?

La presente architettura può essere definita come una **Layered Architecture** arricchita dagli elementi dei design pattern MVC e Facade. Grazie alla netta separazione degli strati, ogni layer ha un ruolo ben definito, contribuendo a rendere il codice più modulare e facilmente manutenibile.

- **Separazione dei compiti:** In particolare, l'uso del layer **Facade** e del layer **Service** consente di isolare il Controller dalla logica di business. Il Controller si occupa esclusivamente di gestire gli input dell'utente e di comunicare con il layer Facade, il quale, a sua volta, interroga il layer Service per eseguire la logica di business. Questa separazione favorisce la scalabilità dell'architettura, rendendola pronta all'integrazione di nuove funzionalità senza richiedere modifiche ai componenti esistenti.
- **Miglior gestione degli errori:** Con questa architettura vi è anche una migliore gestione degli errori andandoli a centralizzare nel Facade che si occuperà di tradurre errori tecnici dovuti alle chiamate con le API esterne o semplicemente instradare dati verso il controller.
- **Facilità di adozione di nuove tecnologie:** Se decido di integrare nuove funzionalità, come il supporto a un'interfaccia web o mobile, potrei mantenere lo stesso Facade e Service, riducendo il lavoro necessario per l'adattamento.

## 10.2 Server

Il server è stato sviluppato utilizzando **Jakarta RESTful Web Services**, un framework Java che fornisce un'API per la creazione di servizi web RESTful. REST (*Representational State Transfer*) è uno stile architettonico per la progettazione di servizi web basato sui principi fondamentali della trasmissione dei dati tramite il protocollo HTTP.

L'architettura del server segue un pattern ben strutturato basato su una variante della **Three-Tier Architecture** e sulla separazione delle responsabilità. In particolare, si distinguono tre livelli principali:

- **Presentation Layer (Controller):** rappresenta il punto di ingresso dell'applicazione. Ha il compito di ricevere le richieste HTTP, validare i dati in input, delegare l'elaborazione al **Service Layer** e restituire una risposta HTTP appropriata.
- **Business Logic Layer (Service):** costituisce il cuore dell'applicazione. Si occupa della logica di business, interagisce con il **DAO** e con altri servizi, gestendo eventuali eccezioni o errori.
- **Data Access Layer (DAO):** è il livello responsabile della persistenza dei dati. Si occupa dell'interazione con il database ed esegue le operazioni **CRUD** (Create, Read, Update, Delete).

## Perché questa architettura?

La seguente architettura può essere definita come una variante dell' archittettura a tre livelli. La Three-Tier Architecture viene utilizzata principalmente per garantire modularità, scalabilità e manutenibilità nelle applicazioni. In particolare questa architettura garantisce :

- **Separazione delle Responsabilità:** Ogni livello ha un ruolo ben definito e svolge un compito specifico, questa separazione rende il codice più leggibile e mantenibile
- **Maggiore manutenibilità:** Se è necessario modificare un livello (es. cambiare il database da MySQL a PostgreSQL), si può farlo senza impattare gli altri livelli.
- **Facilità di Testing:** Possiamo testare il Service Layer senza bisogno di un database reale, grazie all'uso di mock o repository in memoria.

Questa architettura si integra bene con sistemi REST perché separa chiaramente la logica di business dall'interfaccia utente e dal livello di persistenza.

### 10.3 Struttura Schematizzata Back-End



Con questo schema viene riassunto il modo di comunicare fra i vari componenti del Back-End, rendendo la visione del suo funzionamento più semplice e chiara.

## 10.4 Tecnologie Adottate

In questa sezione, descriviamo le tecnologie, i framework e gli strumenti utilizzati per lo sviluppo del software.

## 10.5 JavaFX

JavaFX è un framework di sviluppo per la creazione di interfacce grafiche (GUI) in Java.



Logo di JavaFx

Abbiamo deciso di sviluppare un'applicazione desktop performante ed efficiente, e grazie alle tecnologie avanzate rispetto a quelle più dorate come Swing o AWT, JavaFX è diventata la nostra scelta principale per un'applicazione desktop in grado di soddisfare i requisiti del cliente.

JavaFx È stato progettato per sostituire Swing come toolkit standard per l'interfaccia grafica di Java, offrendo un approccio più moderno allo sviluppo dell'interfaccia utente, più in particolare:

- **Comunità ed Ecosistema:** JavaFX vanta una comunità dedicata e una vasta gamma di librerie e strumenti che ne migliorano la funzionalità. (Come CalendarFx che abbiamo usato)
- **Supporto e Sviluppo:** JavaFX era inizialmente parte del JDK, ma è stato separato a partire dal JDK 11. Da allora, è stato mantenuto come un progetto separato sotto l'egida di OpenJFX, il che ha permesso uno sviluppo continuo e contributi da parte della comunità.
- **Funzionalità Avanzate per l'Interfaccia Utente:** JavaFX offre un insieme completo di controlli per l'interfaccia utente, un supporto al CSS (che in Swing non era presente) e supporto per animazioni e grafica, rendendolo adatto alla creazione di applicazioni visivamente accattivanti.
- **FXML** JavaFX supporta gli FXML, un linguaggio basato su XML che permette agli sviluppatori di definire interfacce utente non scrivendo codice nel sorgente e di conseguenza rendere il codice più pulito e leggibile.

Tuttavia, anche se la sua popolarità è stata in parte oscurata da altri framework come i framework JavaScript (ad esempio, React, Angular) per le applicazioni web e strumenti per lo sviluppo mobile come Flutter e React Native. JavaFX rimane comunque una scelta valida per lo sviluppo di applicazioni desktop in Java, grazie alla sua integrazione con il linguaggio e alle potenti capacità grafiche, come il supporto per animazioni, effetti visivi e la gestione avanzata dei controlli dell'interfaccia utente.

## 10.6 Jakarta Enterprise Edition

Jakarta Enterprise Edition è un'evoluzione di JavaEE, è un insieme di specifiche che estende Java SE con specifiche per funzionalità aziendali come il calcolo distribuito e i servizi web. Le applicazioni Jakarta EE vengono eseguite su runtime di riferimento, che possono essere microservizi o server di applicazioni, i quali gestiscono transazioni, sicurezza, scalabilità, concorrenza e la gestione dei componenti che stanno distribuendo. Jakarta EE include diverse specifiche che servono a scopi differenti, come la generazione di pagine web, la lettura e scrittura da un database in modo transazionale, e la gestione di code distribuite.

Le API di Jakarta EE comprendono diverse tecnologie che estendono le funzionalità delle API di base di Java SE, come Jakarta Enterprise Beans, connettori, servlet, Jakarta Server Pages e diverse tecnologie per i servizi web come **Jax-RS**. **Jakarta RESTful Web Services**, e le sue implementazioni. Jakarta RESTful Web Services è una specifica dell'API Jakarta EE che fornisce supporto nella creazione di servizi web secondo il pattern architettonico Representational State Transfer (REST). JAX-RS utilizza annotazioni, introdotte in Java SE 5, per semplificare lo sviluppo e il deployment di client e endpoint dei servizi web.



Logo di Jakarta EE

### Perchè Jakarta RESTful Web Services?

Jakarta RESTful Web Services è ampiamente utilizzato per creare API REST in applicazioni Java, grazie alla sua semplicità, flessibilità e integrazione con l'ecosistema Java.

- **Standardizzazione e Portabilità:** JAX-RS è un'API, quindi possiamo scegliere tra diverse implementazioni (ad esempio, Jersey, RESTEasy, Apache CXF) senza dover riscrivere il codice.
- **Semplicità e leggibilità:** JAX-RS utilizza annotazioni come @Path, @GET, @POST, @PUT, @DELETE, @Consumes, e @Produces per definire risorse e metodi HTTP in modo chiaro e conciso. Questo riduce la quantità di codice boilerplate e rende il codice più leggibile.
- **Standardizzazione:** Essendo una specifica standard definita dalla piattaforma Jakarta EE, JAX-RS garantisce interoperabilità e portabilità tra diverse implementazioni e server applicativi.

Le più note implementazioni di JAX-RS sono frameworks come

- **Eclipse Jersey** è una delle principali implementazioni di JAX-RS ed è stato scelto proprio questa implementazione nel nostro software, per la sua alta conformità ad JAX-RS e offre tutte le funzionalità standard per la creazione di servizi RESTful.

## 10.7 SceneBuilder

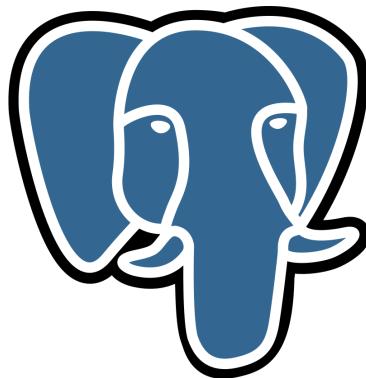
Scene Builder è uno strumento di progettazione visiva per JavaFX che consente agli sviluppatori di creare interfacce utente in modo rapido e intuitivo, evitando la necessità di scrivere manualmente il codice XML per i file FXML. In combinazione con JavaFX, questo software, appositamente sviluppato per tale scopo, permette di progettare interfacce grafiche in modo più efficiente e performante, senza dover redigere manualmente i file FXML.



Logo di SceneBuilder

## 10.8 PostgreSQL

PostgreSQL è una potente base di dati relazionale gratuita ed open source che usa il linguaggio SQL standard ma non disdegnando delle aggiunte utili per conservare e scalare i carichi di dati più complessi (è generalmente definito un "dialetto"). PostgreSQL ha una forte reputazione di RDBMS robusto, affidabile, estensibile. Esso è eseguibile su tutti i principali sistemi operativi e garantisce le proprietà ACID (atomicità, consistenza, isolamento, durabilità) per dati e transazioni. Inoltre fornisce anche utili add-on per estendere le capacità della base di dati, è disponibile sia attraverso riga di comando che GUI



Logo di PostgreSQL

**Perchè PostgreSQL?** PostgreSQL fornisce numerose funzionalità utili per lo sviluppo delle applicazioni, la protezione dell'integrità dei dati e la costruzione di ambienti tolleranti al guasto. Permette anche di scrivere procedure e funzioni procedurali in differenti linguaggi e per differenti linguaggi di programmazione (PL/pgSQL, PL/Tcl, PL/Perl, PL/Python). Tra le funzionalità notevoli, si possono annoverare:

- Supporto ad una vasta quantità di tipi di dato
- Elevata sicurezza
- Supporto sempre gratuito
- Il linguaggio procedurale PL/pgSQL è pienamente compatibile con il linguaggio procedurale di altre basi di dati, facilitando eventuali future migrazioni
- Ampia estensibilità
- Affidabilità e ripresa dal disastro

## 10.9 Docker

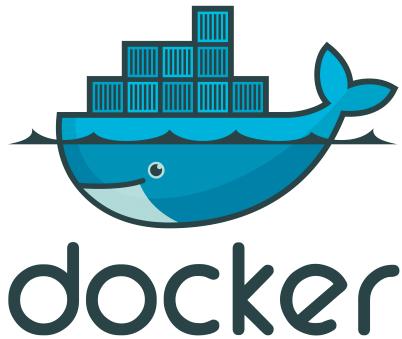
Docker è un insieme di servizi Platform as a Service (PaaS) che utilizza la virtualizzazione a livello di sistema operativo del kernel per eseguire programmi in ambienti isolati e distribuibili chiamati container, i quali sono gestiti dal Docker Engine.

I container sono leggeri e al loro interno conterranno tutto il necessario per l'esecuzione del software, senza preoccuparsi di quale macchina sta eseguendo il Docker Engine.

Questi container sono configurabili attraverso l'uso di file di configurazione appositi.

Compose, inizialmente un add-on per la piattaforma Docker, è stato successivamente incluso al suo interno, e consente attraverso un singolo file YAML la gestione e la comunicazione reciproca di più container.

I container sono un'istanza di un'immagine Docker, ossia un template di sola lettura che istruisce su come il container debba essere creato.



Logo di Docker

### Perchè Docker?

- **Isolamento e Consistenza dell'Ambiente:** Docker ha permesso di isolare il mio software in un container, che include tutte le dipendenze necessarie per eseguirlo su qualsiasi macchina, come librerie, configurazioni di sistema, e il runtime. Questo significa che il mio software può essere eseguito su qualsiasi macchina, senza dover affrontare problemi di compatibilità tra ambienti diversi.
- **Facilità di Configurazione:** Grazie a Docker, abbiamo descritto l'ambiente della nostra applicazione in modo semplice utilizzando un file chiamato Dockerfile. Questo file definisce tutte le istruzioni per costruire il container, come l'installazione delle dipendenze e la configurazione dell'applicazione.

- **Portabilità:** Docker migliora la portabilità del software, consentendo di eseguire l'applicazione su qualsiasi server che supporti Docker, sia esso un server locale, una macchina virtuale o un server nel cloud. Infatti noi dopo aver installato una macchina virtuale su Amazon, abbiamo trasferito il nostro progetto tramite terminale e lo abbiamo avviato utilizzando docker-compose opportunamente configurato. Questo ci ha permesso di esporre il nostro server su un indirizzo pubblico, rendendolo accessibile dall'esterno.

## 10.10 JavaScript

JavaFX fornisce il componente WebView, che permette di visualizzare e interagire con contenuti web all'interno di un'applicazione Java. Grazie al motore WebEngine, è possibile eseguire codice JavaScript e consentire la comunicazione bidirezionale tra Java e JavaScript.

Questa integrazione permette di combinare la potenza di Java con la flessibilità di JavaScript e delle tecnologie web, offrendo un'esperienza utente moderna e interattiva.

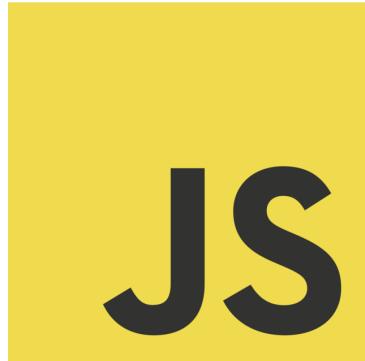


Figure 1: Logo di JavaScript

### 10.10.1 Perchè JavaScript?

Abbiamo integrato JavaScript nel nostro progetto JavaFX per rispondere alle specifiche esigenze del committente. In particolare, l'utilizzo di JavaScript si è reso necessario per garantire un'efficace integrazione con le mappe, funzionalità richiesta nel capitolo.

L'integrazione tra Java e JavaScript è stata impiegata in due contesti principali:

- **1.Creazione dell'annuncio :** l'utente può interagire con la mappa per selezionare la posizione dell'immobile, migliorando l'esperienza d'uso e la precisione nell'inserimento dei dati geografici.
- **2.Ricerca degli annunci :** la mappa consente di visualizzare gli immobili disponibili, offrendo un'interfaccia interattiva e intuitiva per la navigazione e la selezione delle proprietà.

L'adozione di JavaScript ha permesso di combinare la solidità e la struttura di Java con la flessibilità delle tecnologie web, garantendo un'esperienza utente avanzata e conforme ai requisiti del progetto

## 10.11 Altri servizi cloud usati

- **Amazon EC2:** Amazon Elastic Compute Cloud (EC2) è un servizio di cloud computing fornito da Amazon Web Services (AWS) che consente di lanciare e gestire istanze di macchine virtuali. Abbiamo scelto questo servizio perché possiamo facilmente scalare le risorse computazionali in base alle esigenze dell'applicazione. Inoltre, EC2 offre una vasta gamma di tipi di istanze con diverse configurazioni hardware, permettendoci di scegliere quella più adatta alle nostre esigenze di prestazioni e costo.
- **Amazon S3:** Amazon Simple Storage Service (S3) è un servizio di storage di oggetti che consente di archiviare e recuperare dati di qualsiasi dimensione in modo sicuro ed efficiente. Abbiamo utilizzato Amazon S3 per memorizzare immagini e loghi dell'applicazione, poiché offre una durabilità elevata, scalabilità e facilità di integrazione con altri servizi AWS.

The screenshot shows the AWS S3 console. At the top, there are two tabs: "Bucket per uso generico" (selected) and "Bucket di directory". Below the tabs, a banner indicates "Bucket per uso generico (1)" and provides a link to "Tutte le Regioni AWS". A note states: "I bucket sono container per i dati archiviati in S3." A search bar is present. The main table lists one bucket: "bucketde25", located in "Stati Uniti orientali (Virginia settentrionale) us-east-1". It includes a link to "Visualizza l'analizzatore per us-east-1" and a timestamp of "17 Feb 2025 09:10:49 PM CET". Action buttons at the top right include "Copia ARN", "Vuoto", "Elimina", and "Crea bucket". Navigation controls like arrows and a refresh icon are also visible.

Bucket per le immagini

The screenshot shows the AWS EC2 console. At the top, it says "Istanze (1) Informazioni". A search bar and a dropdown menu "Tutti gli st..." are visible. The main table lists one instance: "DietiEstatesSe...", with ID "i-03a375d74789f0093". The status is "Arrestato" (Stopped). Other columns include "Nome", "ID istanza", "Stato dell'istanza", "Tipo di istanza", "Verifica dello stato", "Stato dell'allar", "Zona di disponi...", "DNS IPv4 pubblico", and "Indiriz". Action buttons at the top right include "Connetti", "Stato dell'istanza", "Operazioni", and "Avvia le istanze". Navigation controls like arrows and a refresh icon are also visible.

Istanza EC2

## 11 Motivazione delle scelte di design

### 11.1 Intro



Intro del sistema

Abbiamo deciso di introdurre quando parte l'applicazione un piccolo caricamento da far mostrare all'utente al fine di migliorarne l'usabilità complessiva del prodotto finale e di migliorare il suo coinvolgimento durante l'attesa evitando frustazioni, rendendo l'attesa più piacevole e dare un'identità visiva più curata.

### 11.2 Colori del software

Per il nostro prodotto software abbiamo scelto principalmente due combinazioni di colori: il blu e il bianco, questi colori trasmettono sensazioni ben precise. Il blu perchè dal punto di vista psicologico, ha un effetto rilassante e antistress e si ritiene che stimoli l'inconscio e l'intuizione. Il bianco, invece, rappresenta semplicità e pace. Questa combinazione di colori crea un ambiente visivo che ispira tranquillità, chiarezza e fiducia, elementi fondamentali per un'esperienza utente piacevole ed efficace.

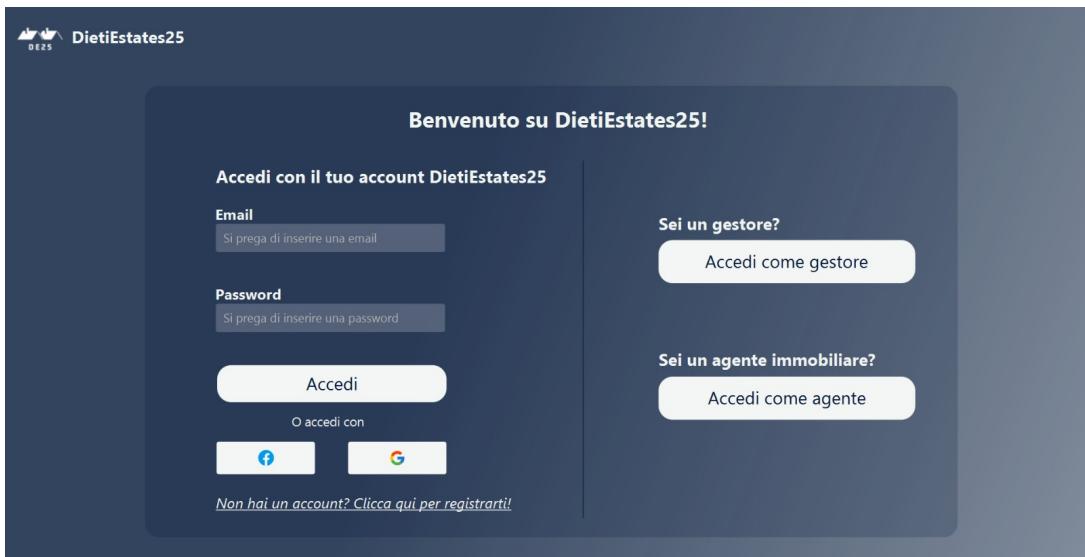
### Color Contrast Checker

Calculate the contrast ratio of text and background colors.

A screenshot of the Color Contrast Checker tool. On the left, there's a light green card with a dark blue border. It contains a text input for 'Text color' with the value '#FFFFFF' and a background input for 'Background color' with the value '#011638'. Below these are two five-star rating icons, one labeled 'Super' and the other '★★★★★'. At the bottom, there are buttons for 'Small text' and 'Large text', each accompanied by three star icons. A note at the bottom says 'Great contrast for all text sizes.' On the right, there's a dark blue card with white text. The title 'Quote n. 21' is at the top, followed by a quote: 'When you go into court you are putting your fate into the hands of twelve people who weren't smart enough to get out of jury duty.' attributed to 'Norm Crosby'. There are also small five-star rating icons at the top of the quote card.

Contrast ratio dei colori principali del sistema

### 11.3 Pagina di benvenuto



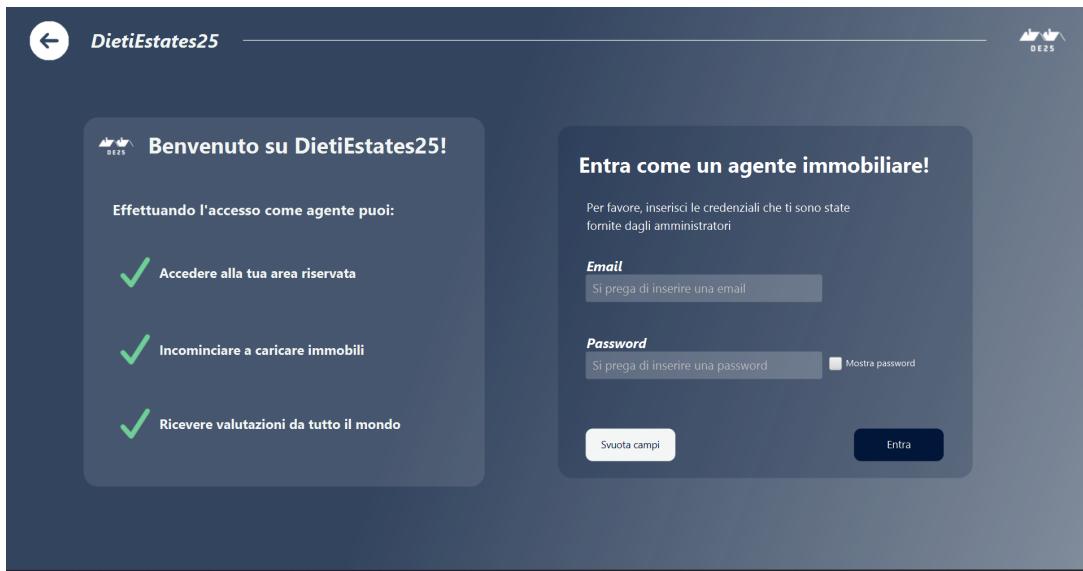
Qui è rappresentata la Home del sistema. All'avvio del software, questa è la prima schermata che viene mostrata all'utente.

Abbiamo scelto di introdurre una pagina di benvenuto invece della classica pagina di ricerca, seguendo le convenzioni delle applicazioni moderne, che spesso presentano un launcher iniziale per orientare l'utente e permettergli di scegliere l'azione da intraprendere. Un'altra motivazione alla base di questa scelta è che non tutti gli utenti del sistema hanno come obiettivo principale la ricerca di un immobile. Ad esempio, gestori e agenti immobiliari potrebbero avere esigenze diverse. Per questo motivo, la Home funge da punto di riferimento chiaro per tutti, guidando ogni tipologia di utente verso la sezione più adatta alle proprie necessità.

### 11.4 Analisi del design

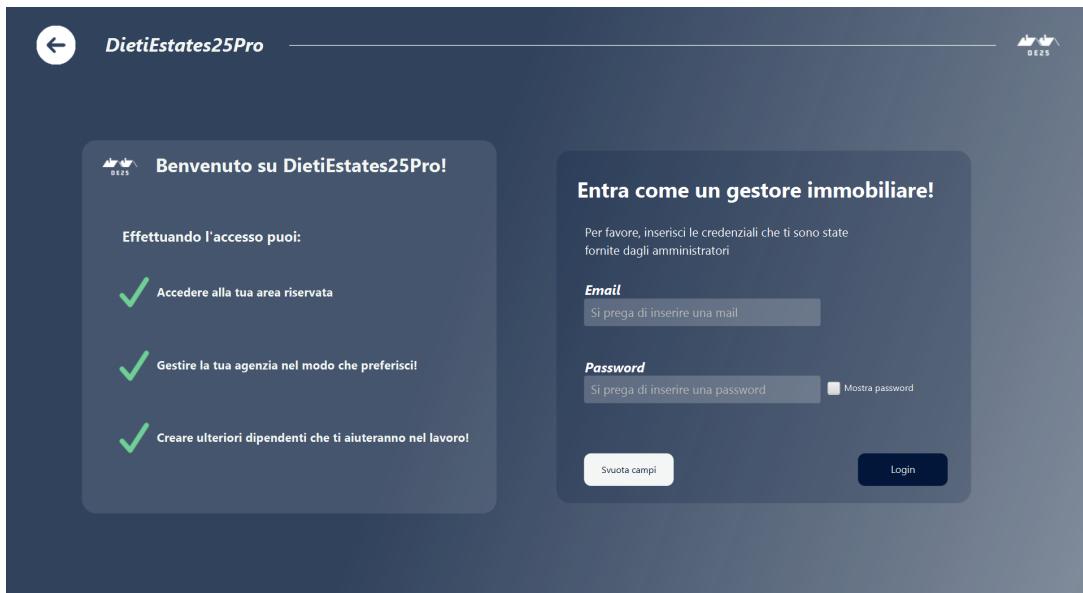
Abbiamo deciso di mantenere una gerarchia visiva semplice nella pagina di benvenuto di DietiEstates25, facendo in modo che i clienti comprendano subito le opzioni di login standard e registrazione. Il pulsante 'Accedi' è posto in evidenza come l'azione principale, facilmente visibile appena si apre la pagina. Gli altri pulsanti, di colore bianco, rappresentano azioni secondarie destinate agli utenti interessati alla parte professionale del sistema. Questi bottoni sono meno evidenti per non distogliere l'attenzione dalla principale azione di login.

## 11.5 Pagine di accesso



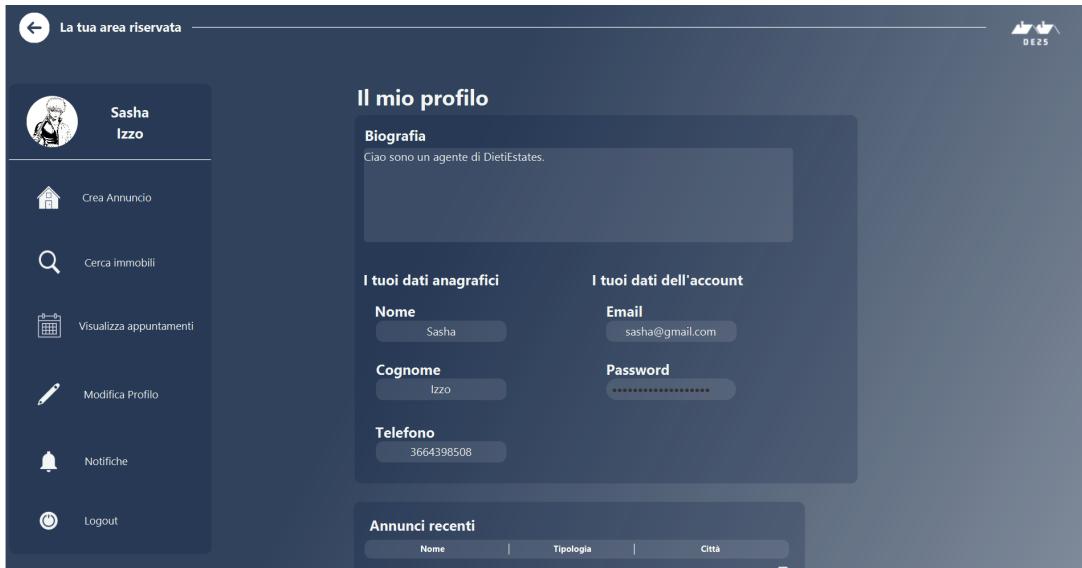
Per la pagine di accesso degli agenti e dei gestori abbiamo utilizzato un design che promuove il **Princípio della regione comune**, al fine di migliorare l'usabilità del prodotto. Grazie a questo principio si delimitano due principali regioni:

- **Regione dei vantaggi** : In questa regione delimitiamo i vantaggi che l'utente ha ad accedere e cosa può fare quando accede al sistema.
- **Regione d'accesso**: Regione dedicata completamente all'accesso dell'agente che spiega che per accedere deve aver ricevuto delle credenziali dagli amministratori.



## 11.6 Aree riservate

Per le aree riservate agli utenti, abbiamo adottato un design che migliora l'usabilità del prodotto finale, basandoci sul **Principio della regione comune** e sul **Principio di prossimità**.

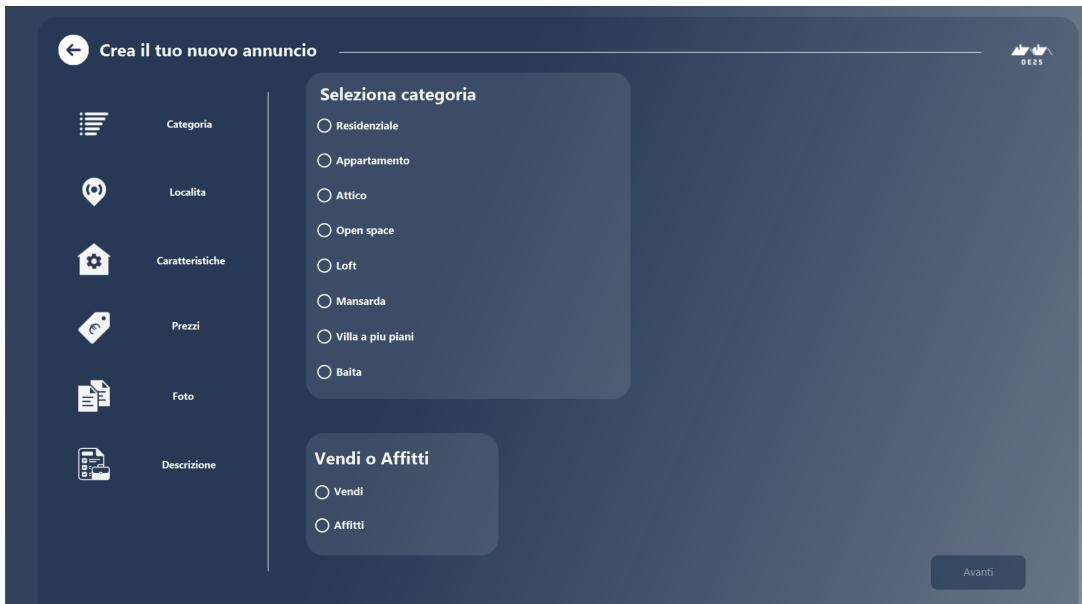


Grazie a questi principi, abbiamo suddiviso la pagina in tre aree principali:

- **Area dati:** È la sezione principale della pagina, in cui vengono mostrati i dati dell'utente, permettendogli di averli sempre sotto controllo. Le informazioni sono organizzate in blocchi distinti e, grazie al **Principio di prossimità**, l'utente può comprendere immediatamente a cosa si riferiscono.
- **Area azioni:** Qui sono raccolte tutte le funzionalità disponibili per l'agente immobiliare. Le operazioni più rilevanti e frequenti sono posizionate in modo da essere facilmente accessibili, riducendo il numero di movimenti necessari con il mouse.
- **Area annunci recenti:** In questa sezione vengono elencati gli annunci più recenti pubblicati dall'agente. Ogni annuncio è chiaramente separato dagli altri per regione al fine di migliorare la leggibilità e l'usabilità del prodotto.

## 11.7 Crea annuncio

Per la pagina di creazione annuncio, abbiamo deciso di adottare un design che minimizzi gli errori, consentendo all'agente di caricare i suoi immobili in modo rapido e piacevole. Abbiamo suddiviso la pagina in diverse aree invece di creare un'unica lunga pagina scrollabile, al fine di migliorare l'esperienza dell'utente. In questo modo, l'agente può cambiare facilmente contesto e sapere immediatamente dove dirigersi, senza dover navigare avanti e indietro nella pagina.



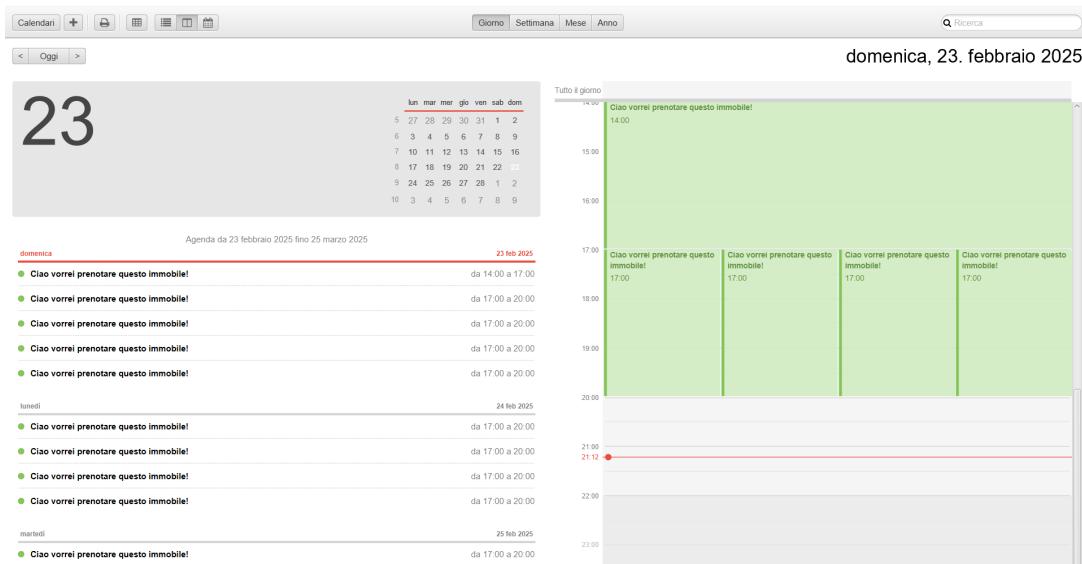
Abbiamo adottato questo design sfruttando il principio della regione comune dove in ogni regione vi sono opzioni correlate fra di loro. Abbiamo anche scelto di minizzare al minimo di errori facendo scegliere l'input all'utente tramite dei radio-button.

Inoltre come possiamo notare sopra in tutte le pagine della creazione dell'annuncio il bottone avanti è bloccato, questo design ha diversi vantaggi:

- **Chiarezza per l'utente:** Gli utenti sanno esattamente cosa devono fare per procedere. Questo evita confusioni e riduce il rischio di errori, in quanto non possono andare oltre senza aver completato tutti i passaggi necessari.
- **Gestione delle aspettative:** Gli utenti sono incentivati a procedere passo per passo, il che crea una percezione di ordine e controllo del processo, riducendo la frustrazione.
- **Riduce frustazione:** Riteniamo che una migliore esperienza utente consista nel permettere al cliente di correggere gli errori man mano che li incontra, piuttosto che ritrovarsi alla fine con errori sparsi in tutta la pagina dopo aver completato il modulo di creazione dell'annuncio. In questo modo, l'utente può risolvere subito ogni problema, evitando frustrazioni e migliorando la fluidità dell'interazione.

## 11.8 Visualizza appuntamenti

Il design della pagina è stato sviluppato tramite la libreria di JavaFx chiamata **JavaFX Calendar**.



L'interfaccia consente di navigare tra gli appuntamenti con facilità, grazie a pulsanti di navigazione per cambiare visualizzazione (giorno, settimana, mese, anno). Inoltre, gli eventi vengono rappresentati con colori distintivi in base alla loro tipologia, facilitando la comprensione a colpo d'occhio. L'utente può interagire con gli appuntamenti tramite clic per visualizzare maggiori dettagli o apportare modifiche. L'integrazione con il database assicura che ogni aggiornamento venga sincronizzato in tempo reale.

## 11.9 Modifica profilo

Il design della pagina di modifica di profilo è stato sviluppato seguendo il **Principio della regione comune** andando a dividere la pagina in due principali sezioni:

- Area visualizzazione:** Questa sezione, situata nella parte sinistra della pagina, è dedicata alla presentazione dell'agente immobiliare. Abbiamo scelto di collocare insieme la sua foto e la biografia perché entrambi gli elementi svolgono un ruolo fondamentale nella comunicazione visiva e nella costruzione della sua identità professionale. La foto permette agli utenti di associare un

volto all'agente, mentre la biografia fornisce informazioni essenziali sul suo percorso e le sue competenze. Unendo questi due aspetti in un'unica area, garantiamo un'esperienza utente più chiara e immediata, agevolando la percezione e la comprensione del profilo dell'agente.

- **Area Dati:** Questa sezione raccoglie tutte le informazioni personali dell'agente, sia di natura anagrafica che relative al suo account. Abbiamo scelto di organizzare i dati in una disposizione verticale per garantire un layout chiaro, ordinato e facilmente leggibile. Questa struttura facilita la consultazione delle informazioni, migliorando l'esperienza utente e rendendo più intuitiva la gestione dei dati.

Abbiamo scelto di rendere i bottoni non cliccabili fino a quando l'utente non effettua una modifica ai propri dati per garantire un'esperienza utente più chiara ed efficiente. Questo approccio presenta diversi vantaggi

- **Migliora la chiarezza dell'interfaccia:** evita che l'utente prema accidentalmente un pulsante senza aver apportato modifiche, riducendo possibili confusioni.
- **Previene azioni inutili:** impedisce l'invio di dati identici a quelli già registrati, evitando richieste superflue al sistema.
- **Guida l'utente nel processo di modifica:** evidenziando l'attivazione dei bottoni solo dopo una modifica, l'interfaccia comunica in modo intuitivo quando un'azione è necessaria.

## 11.10 Creazione account

Per il design della pagina di creazione degli account abbiamo sfruttato il **Principio della regione comune** dividendo la pagina in due principali regioni:

Crea i tuoi agenti immobiliari!

1.Inserisci i dati anagrafici dell'agente

Nome  
Es.Giuliano

Cognome  
Es.Savino

Codice fiscale  
Es.SVNGLN03T05H892R

Prefisso      Telefono  
+39      Es.3664398508

2.Inserisci i dati per il suo account

Email  
Es.email@gmail.com

Password (Almeno 8 caratteri)  
Es.La mia password

Ripulisca campi

Crea

- **Regione dei vantaggi:** In questa regione sono indicati tutti i vantaggi e le conseguenze che l'amministratore avrà quando creerà l'account per i gestori.

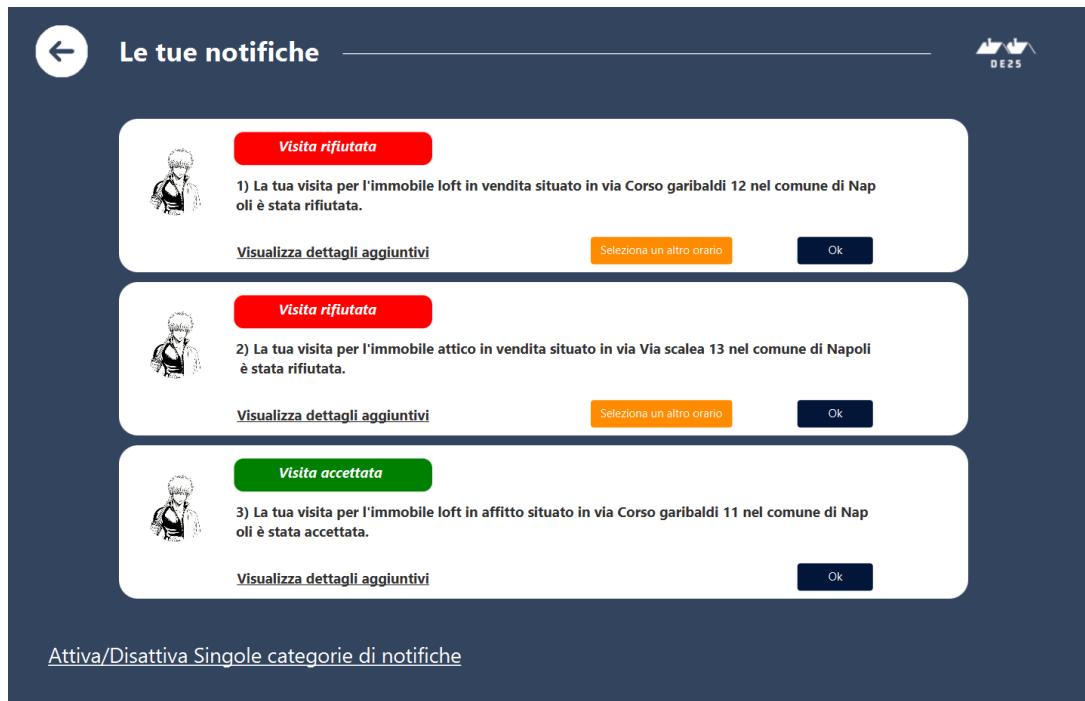
- **Regione dei dati:** In questa regione l'amministratore immetterà i dati per la creazione dell'account. Abbiamo scelto di organizzare i dati in una disposizione verticale per garantire un layout chiaro, ordinato e facilmente consultabile. Questa struttura facilita la consultazione delle informazioni, migliorando l'esperienza utente e rendendo più intuitiva la gestione dei dati.

In questo design abbiamo deciso di mantenere il pulsante "Crea" sempre disponibile e visibile, in questo modo l'amministratore ha un maggiore controllo sull'operazione e può decidere liberamente quando procedere. Questo gli permette di rivedere i dati inseriti in qualsiasi momento, correggere eventuali errori senza restrizioni e avere un riscontro immediato sulla compilazione dei campi.

Inoltre, la visibilità costante del pulsante riduce il rischio di frustrazione, evitando che l'utente debba completare tutti i campi prima di rendersi conto di eventuali problemi. Questo approccio favorisce un'interazione più fluida, intuitiva e consapevole, migliorando l'esperienza utente e riducendo potenziali errori.

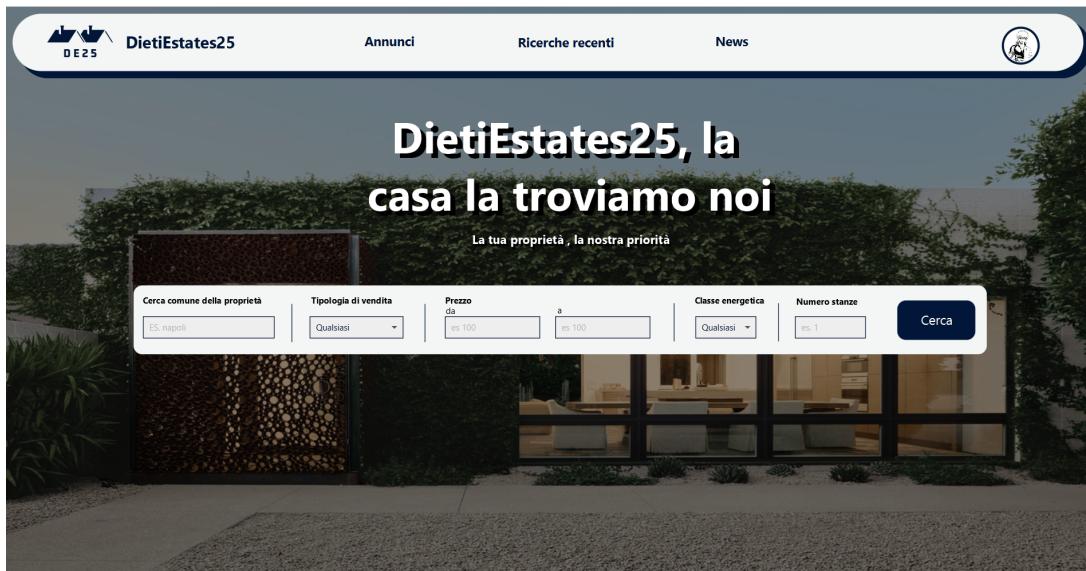
## 11.11 Visualizza notifiche

Per il design della visualizzazione delle notifiche, abbiamo adottato il **Principio della regione comune**, suddividendo ogni notifica in aree distinte, ciascuna con i propri dati e un colore specifico. Questa scelta permette all'utente di riconoscere immediatamente la tipologia di notifica, migliorando la chiarezza e la fruibilità dell'interfaccia



## 11.12 Home di DietiEstates25

Per la home di DietiEstates25, abbiamo adottato un design che minimizza la complessità della pagina, rendendola immediatamente chiara e intuitiva. Una scritta in primo piano comunica in modo diretto lo scopo della schermata, evitando ambiguità. La barra di ricerca è ben visibile fin da subito, senza che l'utente debba navigare attraverso menù complessi, facilitando così l'inizio della ricerca di un immobile. Inoltre, la gerarchia visiva è studiata per guidare l'utente, evidenziando il pulsante "Cerca" come azione principale, mentre le opzioni secondarie sono presentate con un design meno prominente ma comunque accessibile.



## 11.13 Ricerche passate

Per il design delle ricerche passate abbiamo deciso di strutturare ogni ricerca in card diverse questo comporta diversi vantaggi:

- **Maggiore leggibilità:** Le informazioni vengono organizzate in blocchi chiari e distinti, evitando il sovraccarico visivo.
- **Navigazione intuitiva:** Ogni card rappresenta un'entità autonoma, facilitando la scansione visiva e la selezione.

- **Facilità di confronto:** Gli utenti possono confrontare le opzioni rapidamente senza perdere il contesto.



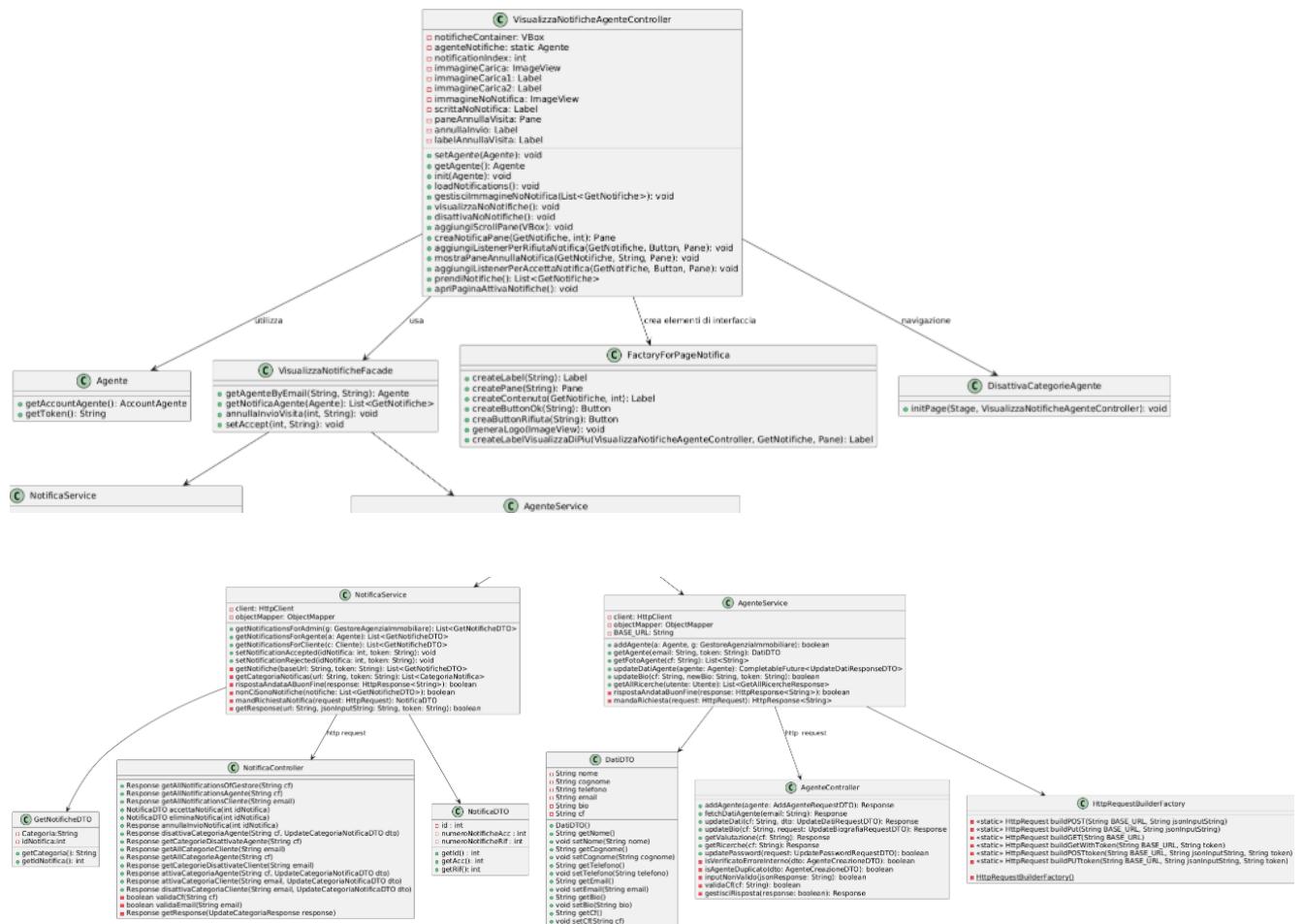
## 12 Diagrammi delle classi di design

Nel processo di sviluppo software, i Diagrammi delle Classi di Design rivestono un ruolo fondamentale nella fase di progettazione, poiché consentono di rappresentare in modo strutturato le classi del sistema, le loro proprietà, i metodi e le relazioni tra di esse. Nei paragrafi successivi verranno illustrati i Diagrammi delle Classi di Design, evidenziandone la struttura, le convenzioni utilizzate e il loro ruolo nell'implementazione del sistema.

## 12.1 Gestione delle notifiche

Il diagramma delle classi presentato rappresenta un sistema per la gestione delle notifiche, probabilmente in un contesto di gestione agenti e comunicazioni tra utenti. Questo schema evidenzia l'interazione tra servizi, controller e Data Transfer Object (DTO) per la gestione delle notifiche.

*Le immagini poste di seguito verranno mostrate divise al fine di migliorare la leggibilità.*



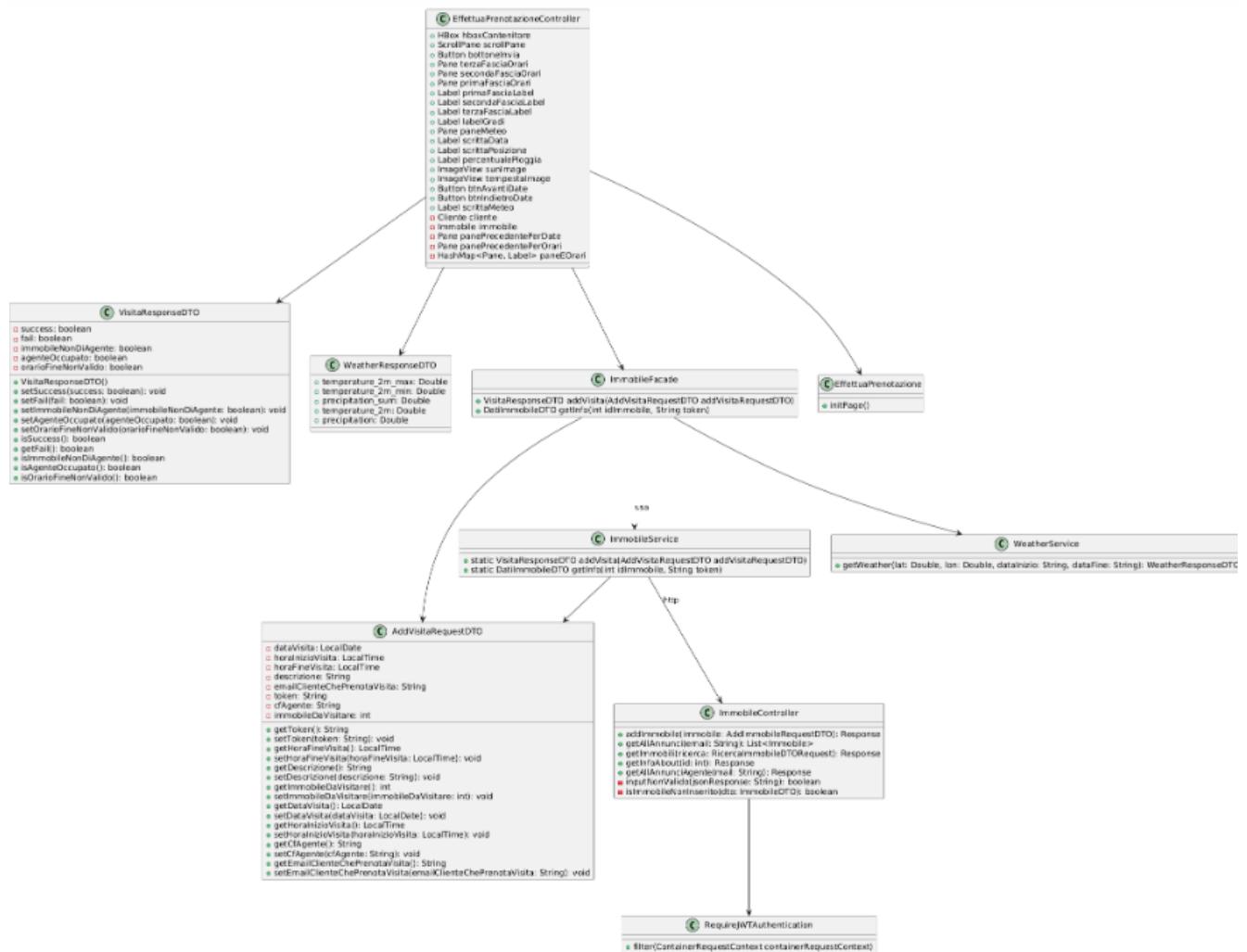
Le principali classi e i loro ruoli sono i seguenti:

- **Agente**: rappresenta un utente con metodi per ottenere le notifiche associate.
  - **NotificaService**: fornisce metodi per recuperare e gestire notifiche.

- **NotificaController**: gestisce le richieste del frontend, permettendo operazioni sulle notifiche.
- **NotificaDTO** e **GetNotificheDTO**: strutture dati per il trasferimento di informazioni sulle notifiche.
- **AgenteService** e **AgenteController**: gestiscono operazioni sugli agenti, come aggiornamenti e recupero dati.
- **FactoryForPageNotifica**: si occupa della creazione di elementi UI per la visualizzazione delle notifiche.
- **VisualizzaNotificheFacade**: funge da intermediario per la gestione delle notifiche e l'interfaccia grafica.
- **HttpRequestBuilderFactory**: classe utility per costruire richieste HTTP.

## 12.2 Effettua prenotazione

Il diagramma delle classi presentato rappresenta un sistema per la gestione di immobili, integrato con un servizio di previsioni meteo. Il sistema è strutturato in modo modulare, con una chiara separazione tra controller, servizi e DTO per la gestione delle richieste e dei dati.



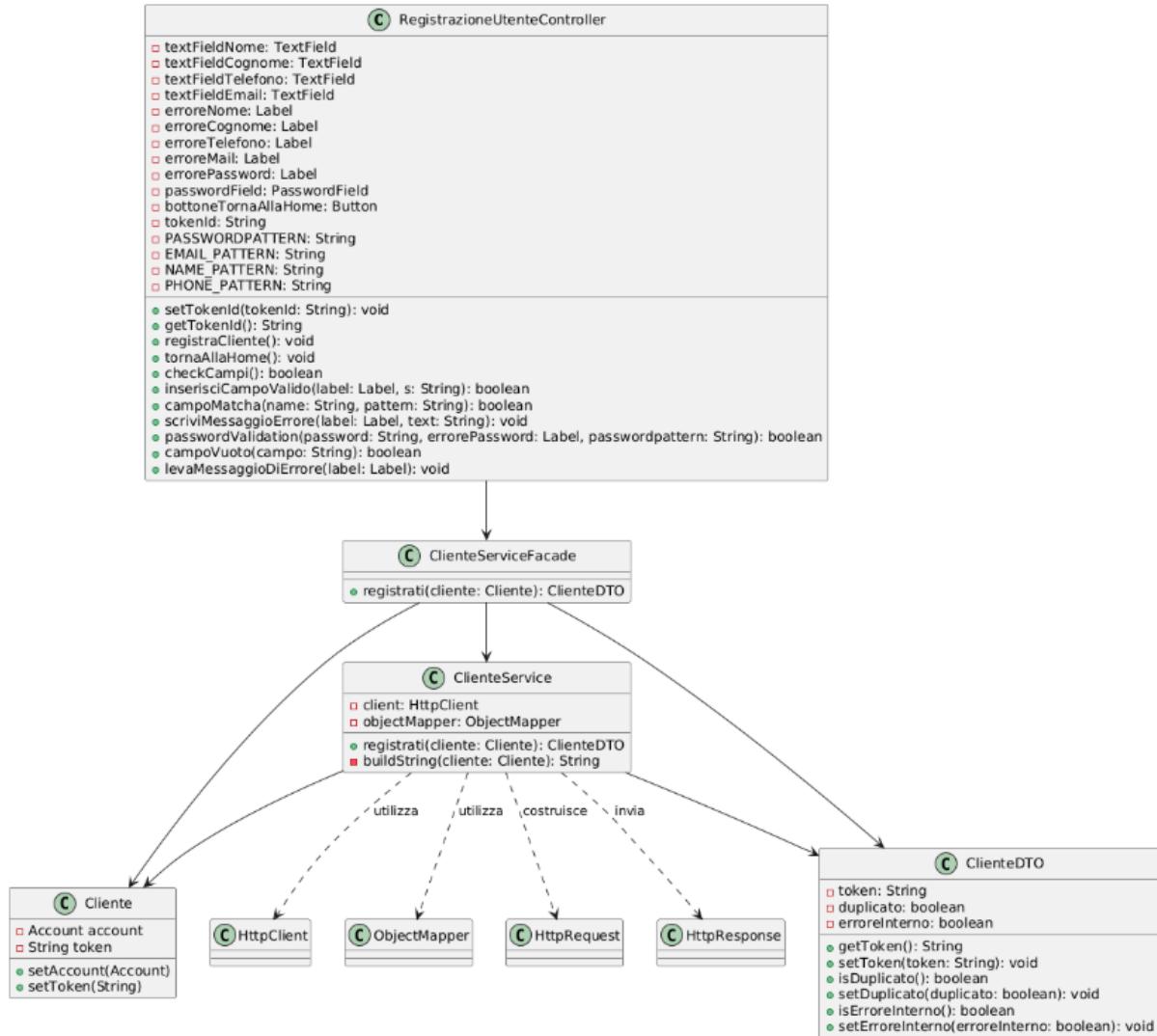
Le principali classi e i loro ruoli sono i seguenti:

- **EffettuaPrenotazioneController**: Classe principale per la gestione delle prenotazioni, contenente vari attributi e metodi per interagire con l'interfaccia utente.
  - **EffettuaPrenotazione**: Fornisce il metodo `initPage()` per inizializzare la vista di prenotazione.

- **ImmobiliFacade**: Funge da intermediario per la gestione degli immobili, offrendo metodi per aggiungere e ottenere informazioni sugli immobili.
- **ImmobiliService**: Implementa i metodi di business logic relativi agli immobili, tra cui il recupero dei dati e la gestione delle prenotazioni.
- **ImmobiliController**: Espone le operazioni per l'aggiunta e la gestione degli immobili tramite API REST, restituendo risposte HTTP.
- **WeatherService**: Classe dedicata al recupero delle previsioni meteo per un determinato immobile, utilizzando latitudine, longitudine e data.
- **WeatherResponseDTO**: DTO che contiene informazioni sulle condizioni meteo.
- **ValutaResponseDTO**: DTO che gestisce l'esito delle operazioni, restituendo lo stato della richiesta.
- **AddValutaResDTO**: DTO con informazioni dettagliate sulle valutazioni degli immobili, incluse coordinate e caratteristiche.
- **RequestJWTAuthentication**: Classe per la gestione dell'autenticazione delle richieste API.

### 12.3 Registrazione

Il diagramma delle classi presentato descrive il sistema di registrazione degli utenti, evidenziando i componenti principali coinvolti nel processo. Il design segue un'architettura modulare, con una netta separazione tra controller, servizi e oggetti di trasferimento dati (DTO).



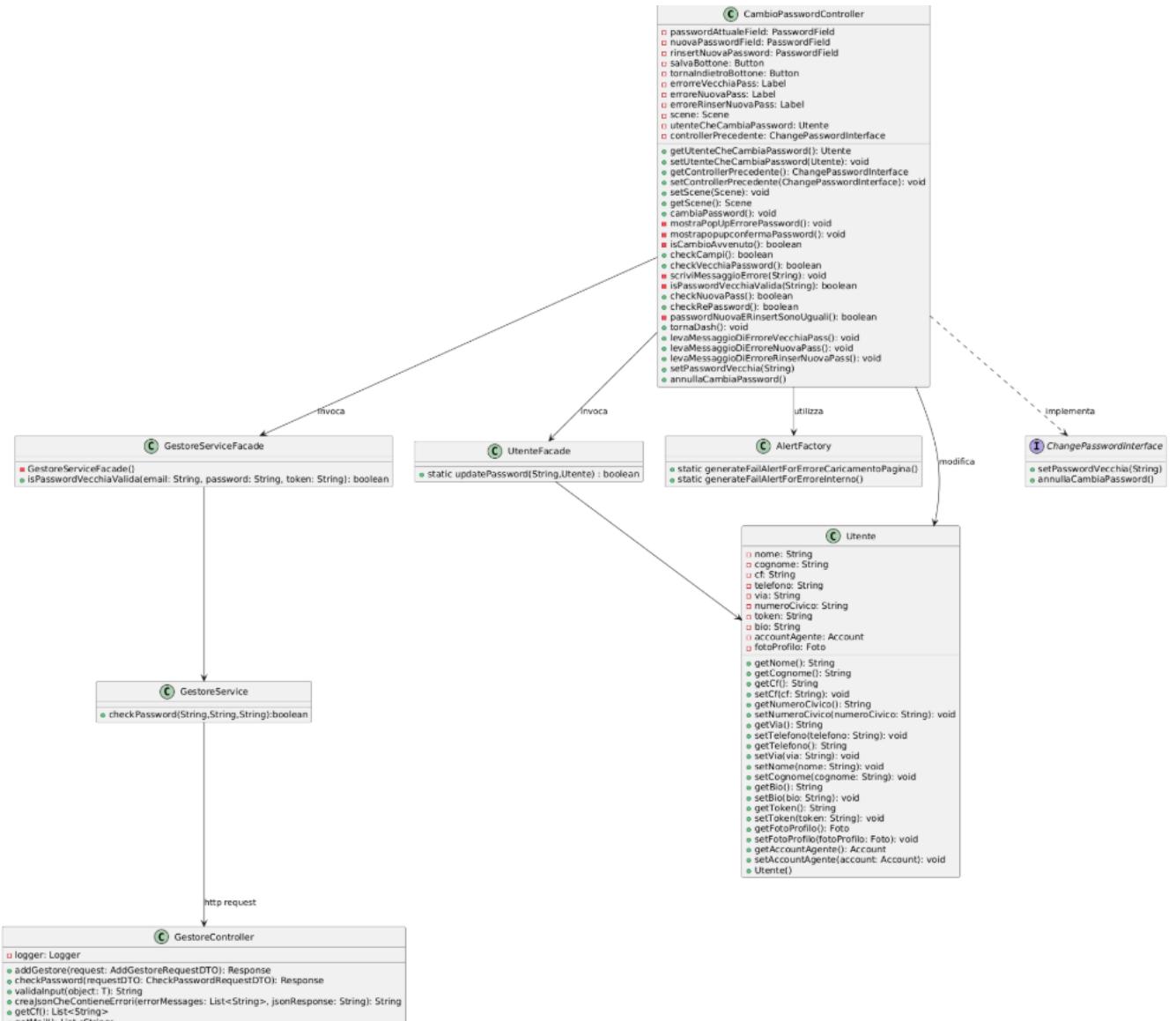
Le principali classi e i loro ruoli sono i seguenti:

- **RegistrazioneUtenteController**: Classe principale che gestisce il processo di registrazione dell'utente, includendo campi di input, validazioni e gestione degli errori.
- **ClienteServiceFacade**: Classe che funge da intermediario tra il controller e i servizi, fornendo un'interfaccia per la registrazione degli utenti.
- **ClienteService**: Implementa la logica di business per la registrazione degli utenti e la costruzione delle richieste.

- **ClienteDTO**: Data Transfer Object che rappresenta i dati del cliente registrato, inclusi token di autenticazione e informazioni di errore.
- **Cliente**: Classe che incapsula le informazioni dell'account utente e il token.
- **HttpClient, ObjectMapper, HttpRequest, HttpResponseMessage**: Classi di supporto utilizzate da **ClienteService** per inviare richieste HTTP e gestire la comunicazione con il backend.

## 12.4 Cambio password

Il presente documento descrive il diagramma delle classi relativo alla gestione del cambio password di un utente. Questo diagramma rappresenta l'architettura software del modulo di autenticazione e sicurezza della piattaforma.



Il diagramma rappresenta diverse classi che collaborano tra loro per implementare il processo di cambio password. Di seguito, una breve descrizione delle principali classi coinvolte:

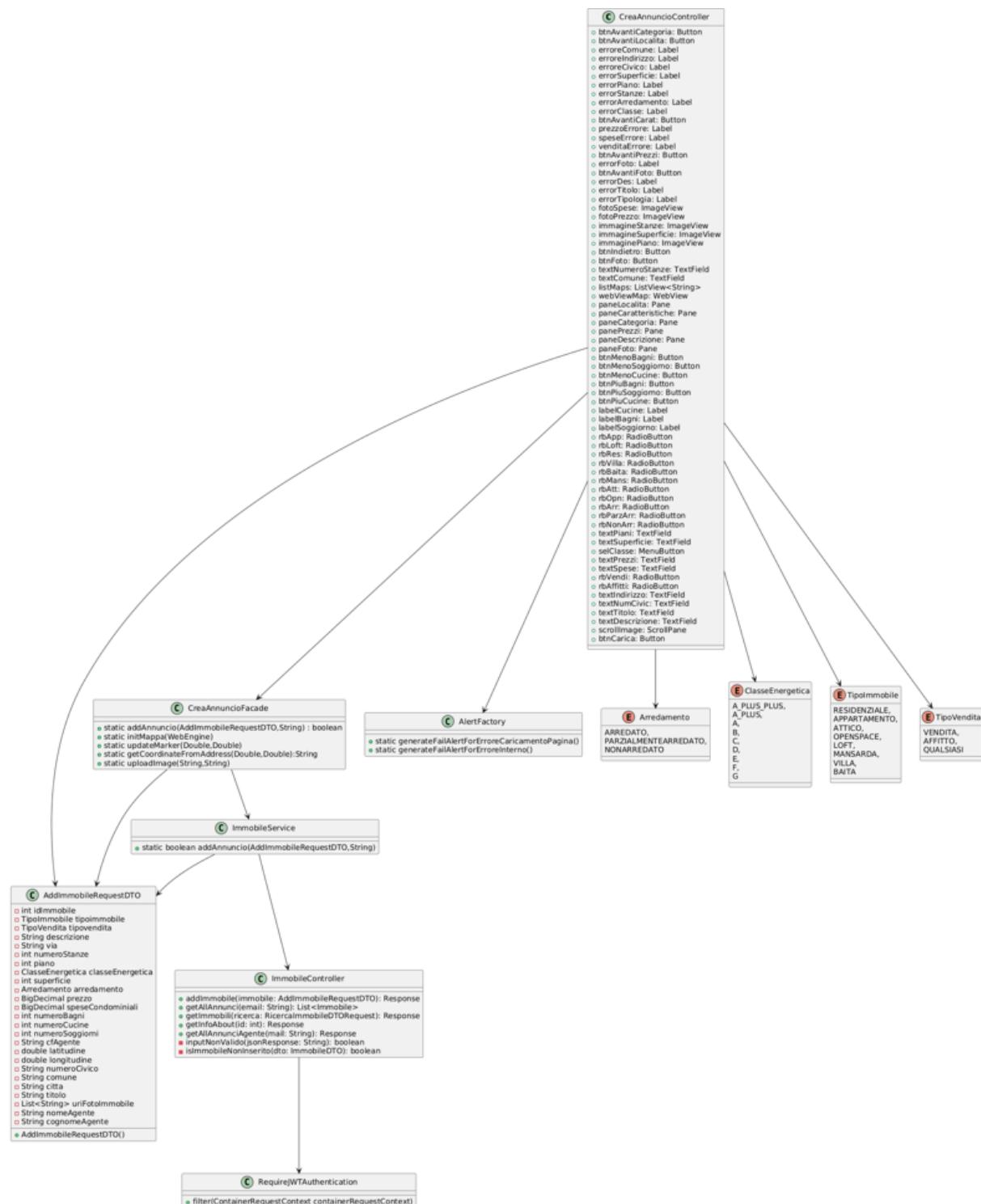
- **CambioPasswordController:** Gestisce l'interfaccia utente per la modifica della password. Contiene campi per la password attuale e quella nuova, pulsanti per confermare o annullare l'operazione e metodi per la validazione degli input.
- **GestoreServiceFacade:** Fornisce un'interfaccia per verificare la validità della password attuale dell'utente prima di consentire la modifica.
- **GestoreService:** Classe di servizio che interagisce con il backend per verificare la password inserita

dall'utente.

- **UtenteFacade:** Permette di aggiornare la password dell'utente dopo la validazione con il servizio di backend.
- **Utente:** Rappresenta l'entità utente con le informazioni personali, tra cui nome, cognome, email, token di autenticazione e password.
- **AlertFactory:** Classe di supporto per generare messaggi di errore o avvisi relativi al processo di cambio password.
- **ChangePasswordInterface:** Interfaccia implementata per astrarre il comportamento del cambio password.

## 12.5 Crea annuncio

Il diagramma delle classi presentato descrive il sistema di gestione degli annunci immobiliari, evidenziando i componenti principali coinvolti nel processo.



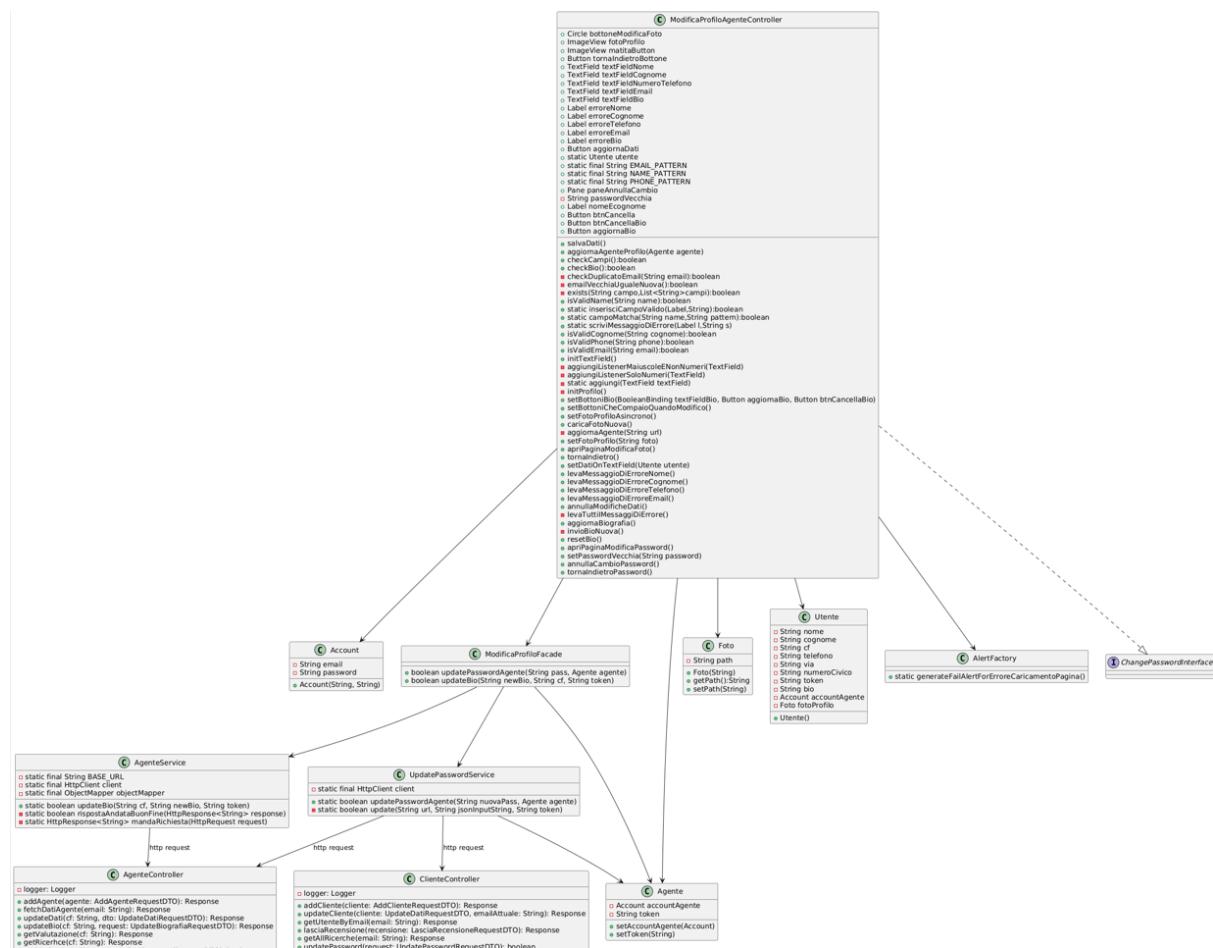
Le principali classi e i loro ruoli sono i seguenti:

- **CreaAnnuncioController**: Classe principale che gestisce il processo di creazione di un annuncio immobiliare, includendo i campi di input, le validazioni e la gestione degli errori.

- **CreaAnnuncioFacade:** Classe che funge da intermediario tra il controller e i servizi, fornendo un'interfaccia per l'inserimento e l'elaborazione degli annunci.
- **ImmobileService:** Implementa la logica di business per la gestione degli annunci immobiliari, inclusa la validazione e il salvataggio.
- **AddImmobilerequestDTO:** Data Transfer Object che rappresenta i dati di un immobile da inserire, comprendendo informazioni come tipo di vendita, classe energetica e descrizione.
- **ImmobileController:** Gestisce le richieste relative agli immobili, offrendo endpoint per l'aggiunta e la consultazione degli annunci.
- **AlertFactory:** Classe di supporto per la generazione di messaggi di errore o conferma nel sistema.
- **Enumerazioni (ClasseEnergetica, TipoImmobile, TipoVendita, Arredamento):** Enum che definiscono le possibili categorie di classe energetica, tipologia di immobile, modalità di vendita e arredamento.
- **RequireJWTAuthentication:** Classe di supporto per la gestione dell'autenticazione JWT, garantendo la sicurezza delle operazioni sugli annunci.

## 12.6 Modifica profilo

Il diagramma delle classi presentato descrive il sistema di modifica del profilo agente, evidenziando i componenti principali coinvolti nel processo.



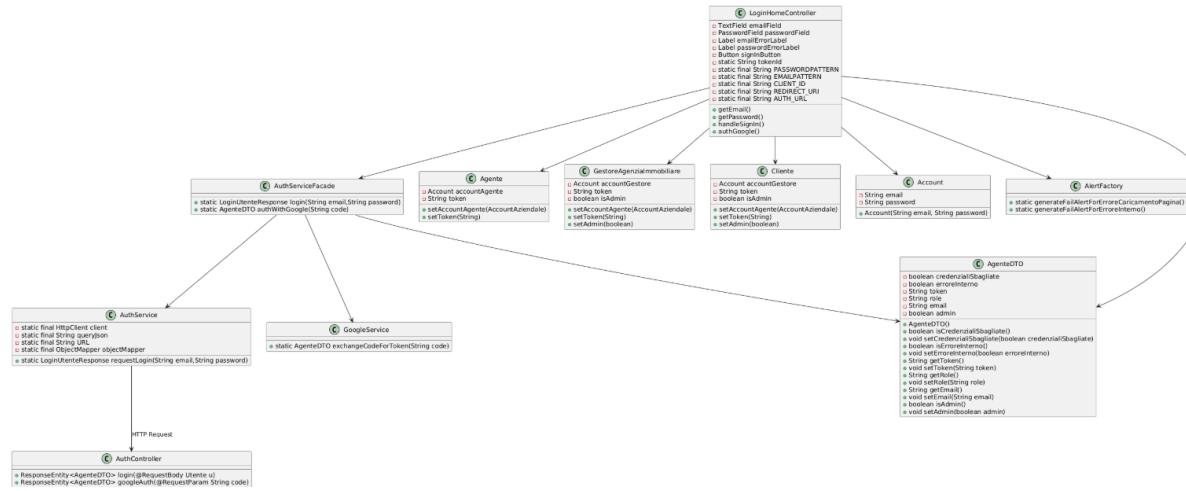
Le principali classi e i loro ruoli sono i seguenti:

- **ModificaProfiloAgenteController**: Classe principale che gestisce il processo di modifica del profilo di un agente, includendo i campi di input, le validazioni e la gestione degli errori.
  - **ModificaProfiloFacade**: Classe che funge da intermediario tra il controller e i servizi, fornendo un’interfaccia per l’aggiornamento dei dati del profilo.
  - **AgenteService**: Implementa la logica di business per la gestione degli agenti immobiliari, inclusa la validazione e l’aggiornamento delle informazioni.
  - **UpdatePasswordService**: Classe dedicata alla gestione del cambio password dell’agente, garantendo la sicurezza dell’operazione.
  - **Account**: Classe che rappresenta l’account di un agente, contenente le credenziali di accesso.
  - **Agente**: Classe che incapsula le informazioni personali e professionali dell’agente immobiliare.

- **Utente:** Classe che rappresenta un utente del sistema, con attributi quali nome, cognome, email e numero di telefono.
- **Foto:** Classe che gestisce le immagini del profilo agente, fornendo metodi per la gestione dei percorsi file.
- **AlertFactory:** Classe di supporto per la generazione di messaggi di errore o conferma nel sistema.
- **ChangePasswordInterface:** Interfaccia per la gestione delle richieste di cambio password.
- **ClientController e AgenteController:** Controller che gestiscono rispettivamente le operazioni relative ai clienti e agli agenti immobiliari.

## 12.7 Autenticazione

Il diagramma delle classi presentato descrive il sistema di autenticazione degli utenti, evidenziando i componenti principali coinvolti nel processo di login e gestione delle credenziali.

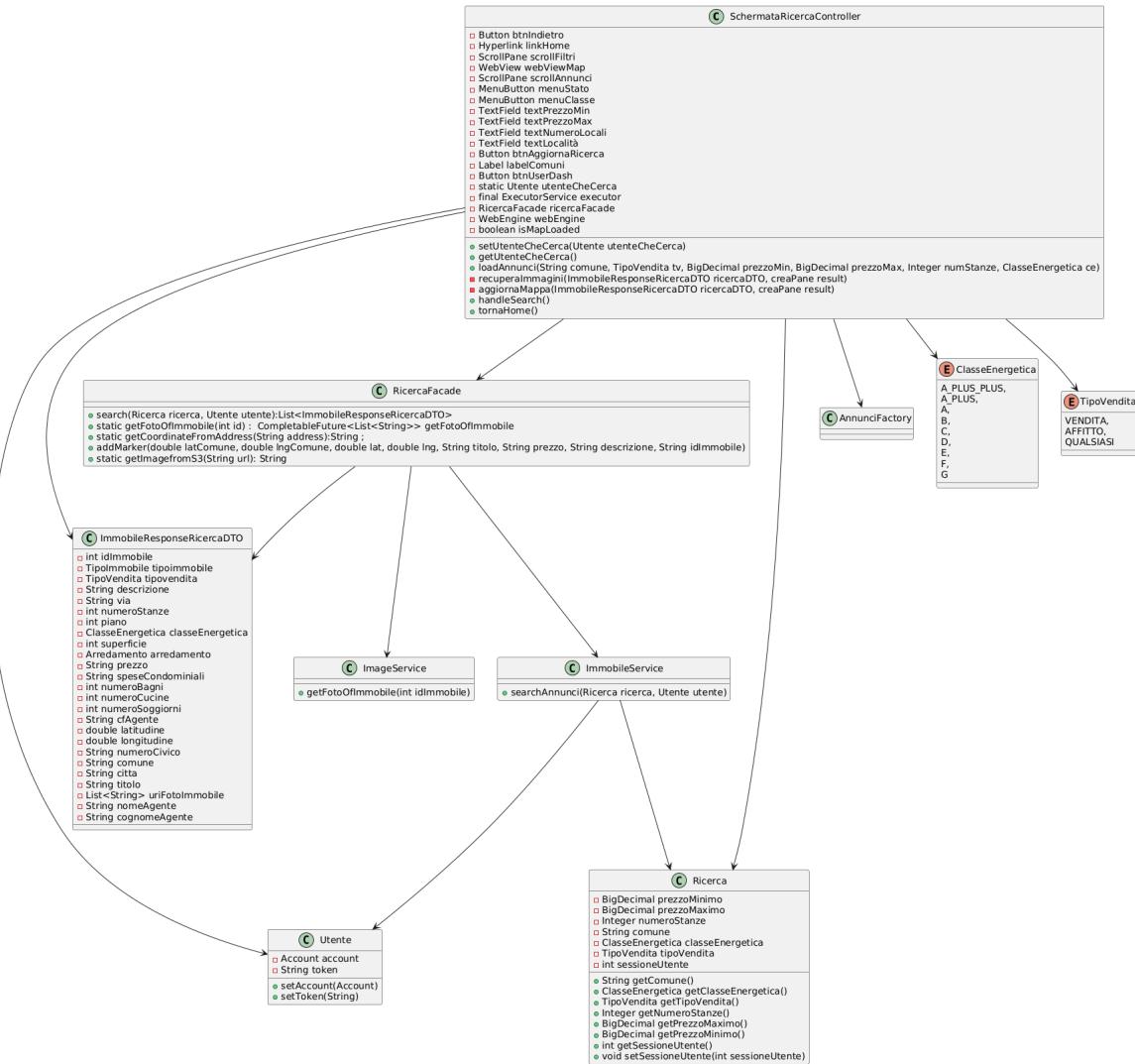


Le principali classi e i loro ruoli sono i seguenti:

- **LoginHomeController**: Classe principale che gestisce il processo di login, incluse le validazioni, la gestione degli errori e l'autenticazione tramite Google.
- **AuthServiceFacade**: Classe che funge da intermediario tra il controller e i servizi, fornendo metodi per il login e l'autenticazione tramite Google.
- **AuthService**: Implementa la logica di business per l'autenticazione degli utenti, interagendo con il backend tramite richieste HTTP.
- **GoogleService**: Gestisce l'autenticazione con Google, scambiando codici di autorizzazione per token di accesso.
- **AuthController**: Controller che gestisce le richieste HTTP per il login e l'autenticazione via Google.
- **Agente, Cliente, GestoreAgenziaImmobiliare**: Classi che rappresentano le diverse tipologie di utenti del sistema, con relative informazioni di account e token di accesso.
- **Account**: Classe che incapsula le credenziali dell'utente, inclusi email e password.
- **AgenteDTO**: Data Transfer Object che rappresenta un agente autenticato, includendo informazioni come credenziali errate, ruolo e token.
- **AlertFactory**: Classe di supporto per la generazione di messaggi di errore nel sistema.

## 12.8 Ricerca

Il diagramma delle classi presentato descrive il sistema di ricerca immobiliare, evidenziando i componenti principali coinvolti nel processo di ricerca e gestione degli annunci.



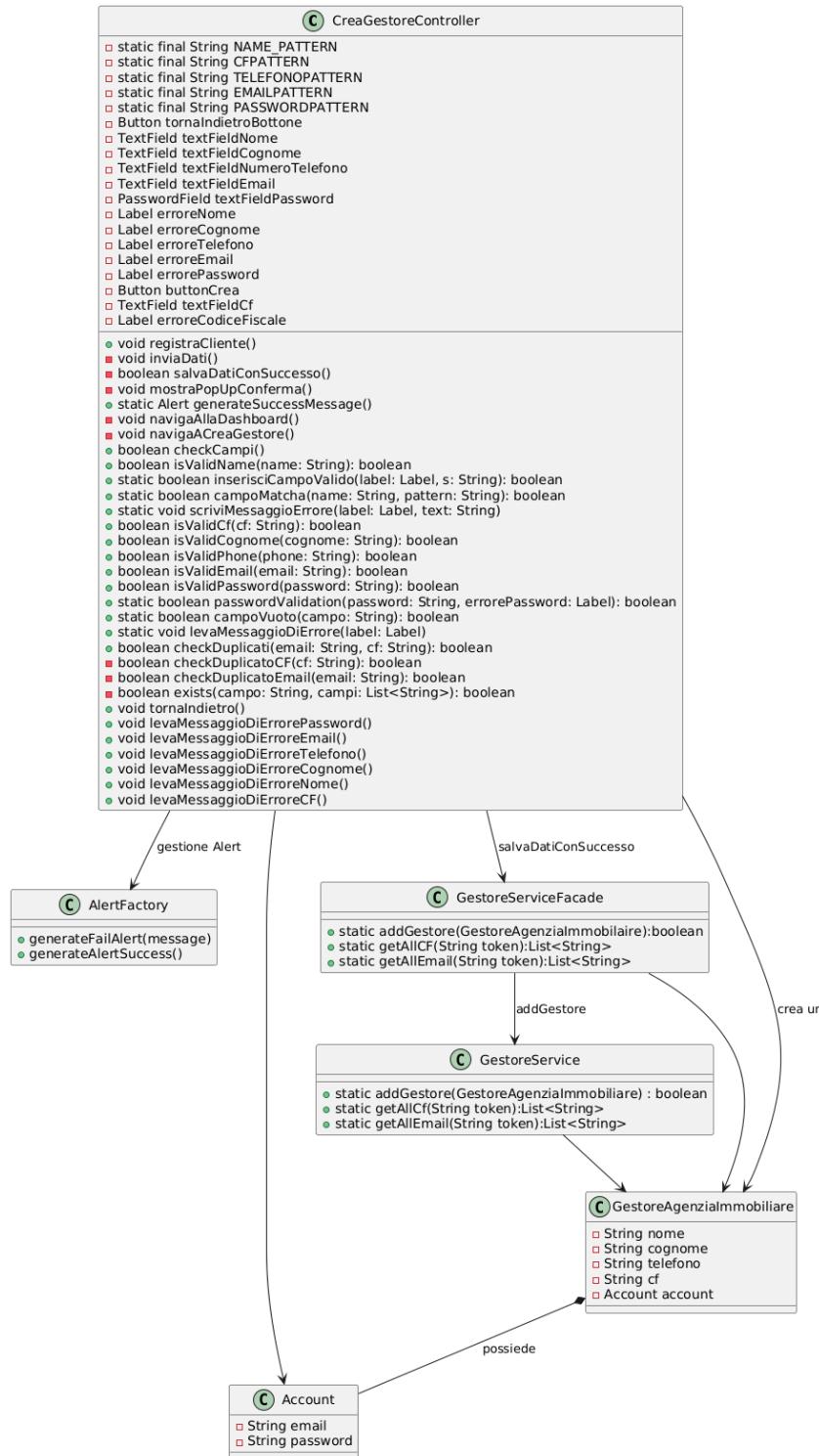
Le principali classi e i loro ruoli sono i seguenti:

- SchermataRicercaController:** Classe principale che gestisce l'interfaccia utente per la ricerca degli immobili, incluse le interazioni con i campi di input e i pulsanti di ricerca.
- RicercaFacade:** Classe che funge da intermediario tra il controller e i servizi, fornendo metodi per ottenere informazioni sugli immobili e aggiungere nuovi annunci.
- ImmobilResponseRicercaDTO:** Data Transfer Object che rappresenta la risposta di una ricerca di immobili, includendo dettagli come descrizione, prezzo e numero di stanze.
- ImmobilService:** Implementa la logica di business per la ricerca e gestione degli annunci immobiliari.
- ImageService:** Classe responsabile della gestione delle immagini associate agli immobili.

- **Utente:** Classe che rappresenta l'utente del sistema, con informazioni di autenticazione e sessione.
- **Ricerca:** Classe che incapsula i criteri di ricerca impostati dall'utente, inclusi intervalli di prezzo, numero di stanze e classe energetica.
- **ClasseEnergetica, TipoVendita:** Classi di supporto che rappresentano rispettivamente la classificazione energetica degli immobili e la tipologia di transazione (vendita, affitto, ecc.).
- **AnnunciFactory:** Classe che fornisce metodi per la generazione e gestione degli annunci immobiliari.

## 12.9 Creazione account

Il diagramma delle classi presentato descrive il sistema di gestione degli agenti e delle credenziali, evidenziando i componenti principali coinvolti nel processo di autenticazione e gestione degli utenti.



Le principali classi e i loro ruoli sono i seguenti:

- **LoginHomeController:** Classe principale che gestisce l'interfaccia utente per l'autenticazione, incluse le interazioni con i campi di input e i pulsanti di login.

- **AuthServiceFacade:** Classe che funge da intermediario tra il controller e i servizi di autenticazione, fornendo metodi per il login degli utenti e l'autenticazione tramite Google.
- **AuthService:** Implementa la logica di business per la gestione dell'autenticazione degli utenti, effettuando richieste HTTP ai servizi di autenticazione.
- **GoogleService:** Classe responsabile dello scambio di codici di autenticazione con Google per ottenere token di accesso.
- **AuthController:** Classe che gestisce le richieste di login e autenticazione degli utenti, esponendo endpoint per l'accesso tramite credenziali o Google.
- **Agente:** Classe che rappresenta un agente immobiliare autenticato nel sistema, contenente riferimenti all'account associato e al token di autenticazione.
- **GestoreAgenziaImmobiliare:** Classe che rappresenta un gestore di agenzia immobiliare, contenente informazioni sull'account e privilegi amministrativi.
- **Cliente:** Classe che rappresenta un cliente registrato nel sistema, contenente dati di autenticazione e ruolo amministrativo.
- **Account:** Classe che incapsula le informazioni di login degli utenti, come email e password.
- **AgenteDTO:** Data Transfer Object che rappresenta un agente autenticato, includendo informazioni sulle credenziali e il ruolo.
- **AlertFactory:** Classe di supporto per la generazione di alert relativi ad errori di autenticazione o problemi interni al sistema.

## 13 Diagramma di sequenza di design

I diagrammi di sequenza (Sequence Diagrams) sono uno strumento essenziale nella modellazione dei sistemi software, in particolare nella fase di analisi e progettazione.

Questi diagrammi sono particolarmente utili per descrivere casi d'uso specifici, il comportamento dei sistemi distribuiti e le interazioni tra componenti software. Forniscono una visione chiara della logica operativa, facilitando la comunicazione tra sviluppatori, analisti e stakeholder.

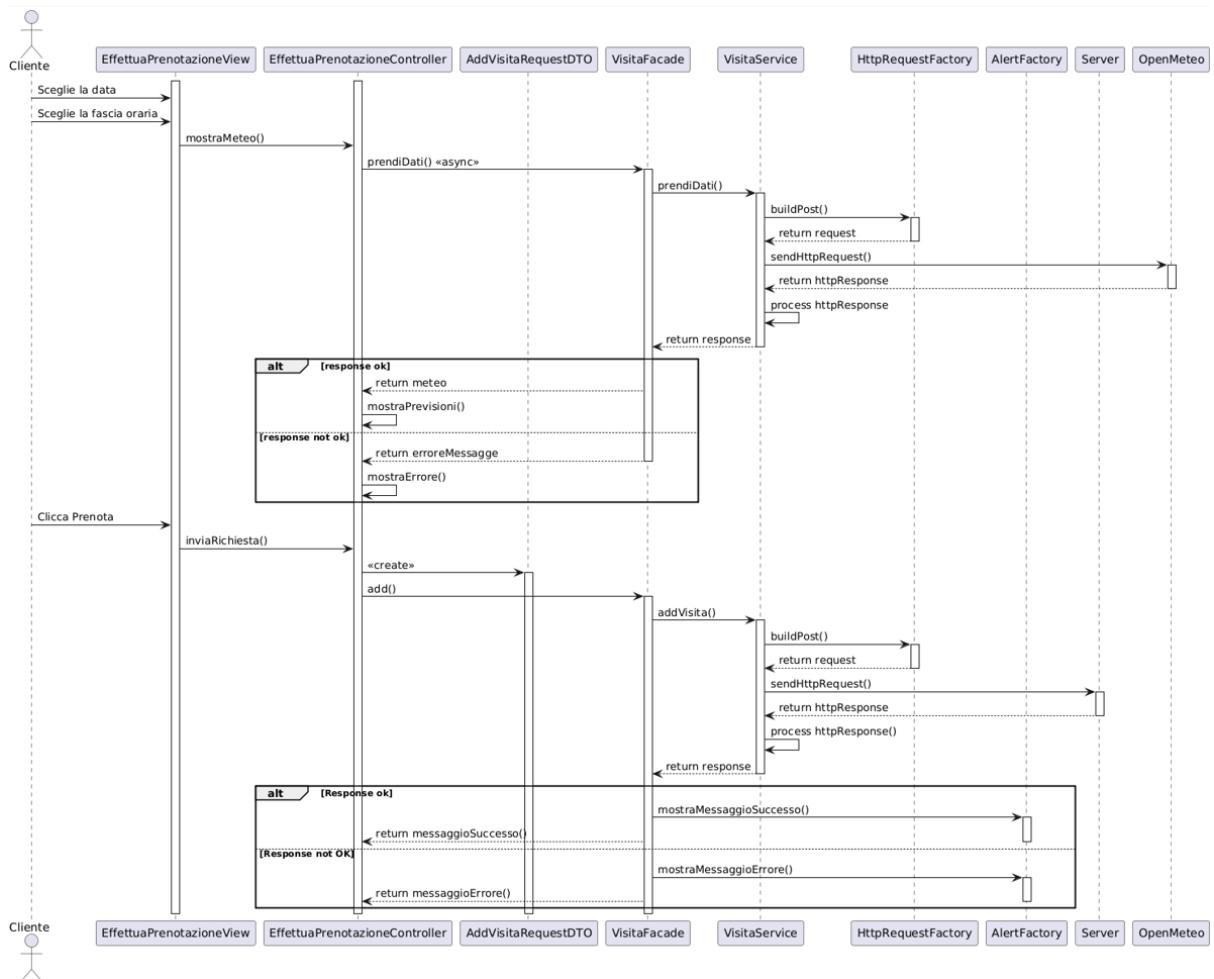
### 13.1 Caso d'uso : Prenota visita

Il seguente diagramma di sequenza illustra il processo di effettuazione di una prenotazione da parte del cliente. L'attore principale, il cliente, seleziona una data tra quelle disponibili e una fascia oraria in cui desidera visitare l'immobile.

A questo punto, il sistema interagisce con un servizio esterno denominato **OpenMeteo** per ottenere le previsioni meteorologiche relative alla data selezionata, le quali vengono successivamente mostrate al cliente.

Successivamente, il cliente procede con la conferma della prenotazione cliccando sul relativo pulsante. Il sistema elabora i dati forniti e invia una richiesta al server per la registrazione della prenotazione.

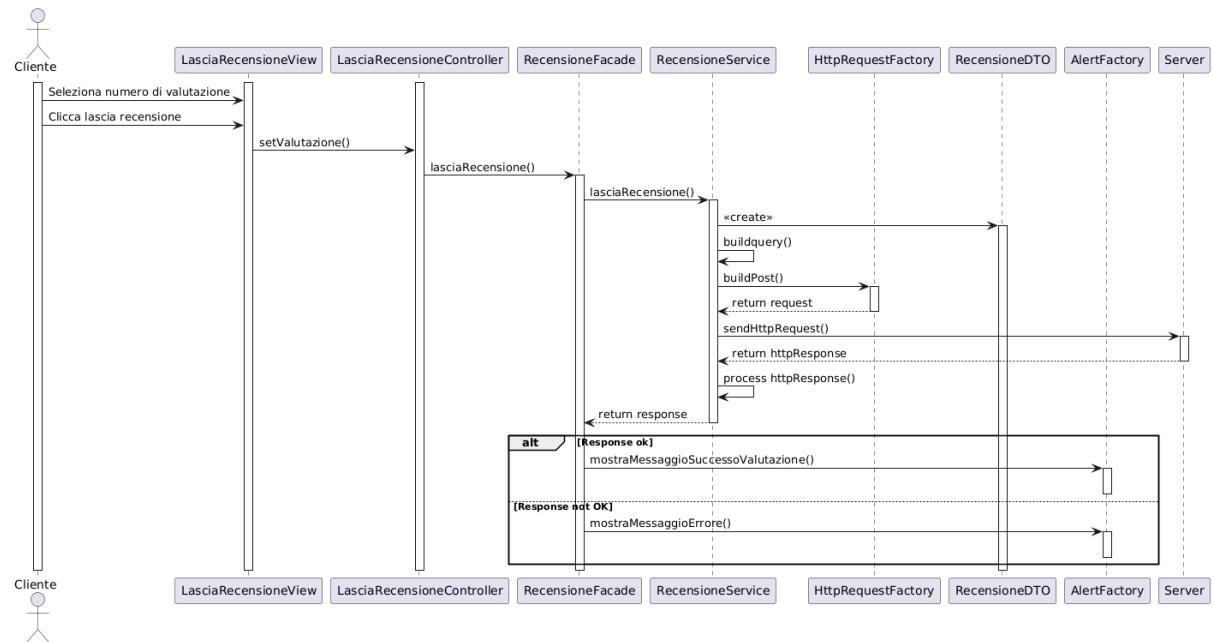
Se la richiesta viene elaborata con successo, il sistema aggiorna lo stato della prenotazione e notifica il cliente mediante un messaggio di conferma visualizzato nell'interfaccia grafica. In caso contrario, il sistema interpreta il messaggio di errore restituito dal server e fornisce al cliente una comunicazione adeguata, descrivendo il problema riscontrato.



## 13.2 Caso d'uso: Lascia recensione

Il seguente diagramma di sequenza illustra il processo di rilascio di una recensione da parte del cliente. L'attore principale, il cliente, seleziona una valutazione da assegnare all'agente e procede cliccando sul pulsante "Invia recensione". Il sistema elabora i dati forniti e invia una richiesta al server per la registrazione della recensione.

Qualora la richiesta venga elaborata con successo, il sistema notifica il cliente mediante un messaggio di conferma visualizzato nell'interfaccia grafica. In caso di errore, il sistema interpreta il messaggio di errore restituito dal server e fornisce al cliente un avviso adeguato, descrivendo il problema riscontrato.



# Artefatti Software

In questa sezione analizziamo il Dockerfile e il processo di build automatica per configurare e distribuire l'applicazione backend in un ambiente Dockerizzato.

## 14 DockerFile e file di build Automatica

In questa sezione analizziamo il Dockerfile e il processo di build automatica per configurare e distribuire l'applicazione backend in un ambiente Dockerizzato.

### 14.1 File di build Automatica

I file di build automatica sono utilizzati per gestire il processo di costruzione di un progetto software in modo automatizzato. Essi definiscono le istruzioni necessarie per compilare, testare, pacchettizzare e distribuire il codice, garantendo coerenza e riproducibilità nel processo di sviluppo.

### 14.2 DockerFile

```
FROM openjdk:21-jdk

COPY target/ProvaProgettoV2-1.0-SNAPSHOT.jar /usr/app/

WORKDIR /usr/app

ENTRYPOINT ["java", "-jar", "ProvaProgettoV2-1.0-SNAPSHOT.jar"]
```

DockerFile Back-end

Il **Dockerfile** definisce l'ambiente di **build** e **runtime** necessario per eseguire un'applicazione Java all'interno di un container Docker. In particolare, questa configurazione costruisce un'immagine basata su **OpenJDK 21**, includendo il file **.jar** dell'applicazione e definendo il processo di avvio.

#### 14.2.1 Struttura del Dockerfile

- **Base Image:** Viene utilizzata l'immagine ufficiale `openjdk:21-jdk`, che fornisce un ambiente Java completo.
- **Copia dell'Applicazione:** Il comando `COPY` trasferisce il file `ProvaProgettoV2-1.0-SNAPSHOT.jar` nella directory `/usr/app/` all'interno del container.
- **Impostazione della Directory di Lavoro:** Il comando `WORKDIR /usr/app` definisce la directory di lavoro, semplificando l'esecuzione dei comandi successivi.

- **Avvio dell’Applicazione:** L’ENTRYPOINT specifica il comando da eseguire quando il container viene avviato, utilizzando `java -jar` per eseguire il file `.jar`.

Questa configurazione garantisce che l’applicazione venga eseguita in modo isolato e riproducibile in qualsiasi ambiente Docker.

#### 14.2.2 Docker compose

```

services:
  dieti-estates-app:
    build: .
    ports:
      - "9094:9094"
    stdin_open: true
    tty: true
    depends_on:
      - database
    environment:
      DATABASE_URL: "jdbc:postgresql://database:5432/DietiEstates"
      DATABASE_USER: "neondb_owner"
      DATABASE_PASSWORD: ${DATABASE_PASSWORD}
      DATABASE_SSL_MODE: "require"

  database:
    image: postgres:15
    restart: always
    environment:
      POSTGRES_DB: DietiEstates
      POSTGRES_USER: neondb_owner
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
    ports:
      - "5432:5432"

```

Docker Compose Back-end

Il file `docker-compose.yml` definisce due servizi principali: l’applicazione DietiEstates e il database PostgreSQL. Il file configura la creazione e la gestione di un’applicazione Docker, che include le variabili di ambiente necessarie per la connessione al database, nonché le impostazioni di rete e di accesso.

#### 14.2.3 6.1.3 Accesso Interattivo al Container

Le opzioni `stdin_open` e `tty` sono abilitate per consentire l’accesso interattivo al container. Questo approccio è particolarmente utile per eseguire operazioni in tempo reale, come il debugging o altre attività di amministrazione direttamente all’interno del container.

- `stdin_open`: Abilita il flusso di input da tastiera (`stdin`) nel container.
- `tty`: Consente di allocare un terminale per l’accesso interattivo.

L'attivazione di queste opzioni permette di interagire direttamente con il container tramite una sessione terminale, facilitando diagnosi rapide e interventi immediati.

#### 14.2.4 Gestione delle Password

Le password, come `DATABASE_PASSWORD` e `POSTGRES_PASSWORD`, sono gestite tramite variabili d'ambiente definite in un file `.env`. In questo modo, le credenziali non sono esposte direttamente nel file di configurazione `docker-compose.yml`, garantendo una maggiore sicurezza. Questa modalità di gestione delle password permette di mantenere le informazioni sensibili al sicuro, evitando che vengano visibili nel codice o nel file di configurazione, riducendo il rischio di esposizione accidentale.

## 15 Strumento di versioning

### 15.1 Scelte adottate

GitHub è una piattaforma di hosting per il controllo di versione e la collaborazione, costruita attorno al sistema di versioning distribuito Git. Questo strumento è ampiamente utilizzato nella comunità degli sviluppatori per gestire progetti software. Per il versioning di **DietiEstates25** abbiamo scelto proprio Github, andando a separare la repository del Client, da quella del Server, per una maggiore organizzazione del progetto.

#### Vantaggi di GitHub come strumento di versioning

L'utilizzo di GitHub come strumento di versioning presenta numerosi vantaggi, tra cui:

- **Collaborazione facilitata:** GitHub consente a più sviluppatori di lavorare contemporaneamente sullo stesso progetto senza conflitti. Attraverso le funzionalità di pull request, gli utenti possono proporre modifiche al codice, che possono essere revisionate e integrate nel progetto principale. Questo processo rende la collaborazione più fluida e organizzata.
- **Tracciamento delle modifiche:** Ogni modifica apportata al codice è registrata nella cronologia del repository. Gli sviluppatori possono visualizzare chi ha apportato una modifica, quando e perché, il che aiuta a mantenere una documentazione chiara delle evoluzioni del progetto.
- **Ritorno a versioni precedenti:** GitHub consente di annullare modifiche o di tornare a versioni precedenti del codice. Questo è particolarmente utile quando una nuova funzionalità introduce un bug o quando è necessario ripristinare un comportamento precedente.
- **Backup e sicurezza :** I repository su GitHub sono ospitati nel cloud, fornendo un backup sicuro del codice.

## 15.2 Repository

```
services:
  dieti-estates-app:
    build: .
    ports:
      - "9094:9094"
    stdin_open: true
    tty: true
    depends_on:
      - database
    environment:
      DATABASE_URL: "jdbc:postgresql://database:5432/DietiEstates"
      DATABASE_USER: "neondb_owner"
      DATABASE_PASSWORD: ${DATABASE_PASSWORD}
      DATABASE_SSL_MODE: "require"

  database:
    image: postgres:15
    restart: always
    environment:
      POSTGRES_DB: DietiEstates
      POSTGRES_USER: neondb_owner
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
    ports:
      - "5432:5432"
```

Repository usata per DietiEstates25

## 16 Report di qualità del codice

La qualità del codice rappresenta un elemento fondamentale nello sviluppo software. Un codice di alta qualità facilita la comprensione e la manutenzione da parte degli sviluppatori, riduce il numero di bug e vulnerabilità, aumentando la stabilità dell'applicazione. Inoltre, un codice ben strutturato e pulito è essenziale per garantire la scalabilità del progetto. Permette di aggiungere nuove funzionalità senza compromettere le performance o l'affidabilità del sistema.

### 16.1 SonarQube

Per monitorare e migliorare la qualità del codice, è stato impiegato SonarQube. Questo permette di identificare potenziali bug, vulnerabilità e problemi tecnici, fornendo suggerimenti per ottimizzare il codice.

- **Duplicazioni** : Le duplicazioni di codice possono portare a una maggiore complessità e difficoltà nella manutenzione. SonarQube rileva segmenti di codice duplicati e suggerisce di refattorizzarli per migliorare l'efficienza e ridurre il rischio di inconsistenze.
- **Bugs** : Rappresentano errori nel codice che possono causare comportamenti imprevisti o malfunzionamenti dell'applicazione.
- **Vulnerabilità** : Questi indici segnalano potenziali rischi di sicurezza presenti nel codice.
- **Code Smells** : Indicano aree del codice che, pur non essendo errori diretti, possono ridurre la leggibilità, la manutenibilità o la performance dell'applicazione.



Report di SonarQube per DietiEstates25

# Testing

## 16.2 Unit Testing

All'interno del progetto, lo Unit Testing è stato implementato seguendo la metodologia **Black Box Testing**. Questo approccio si concentra sulla verifica della correttezza delle funzionalità esposte dai moduli del software senza analizzare la loro implementazione interna. L'obiettivo è garantire che, dati specifici input, il sistema restituisca gli output attesi, verificando il comportamento complessivo dell'applicazione piuttosto che il codice sorgente sottostante.

Per l'implementazione dei test unitari è stato utilizzato il framework JUnit, che adotta il modello xUnit, come richiesto dal committente. Tale modello è ampiamente riconosciuto per la strutturazione e l'automazione dei test, consentendo un'efficace organizzazione delle verifiche e una gestione strutturata dei casi di test.

### 16.2.1 lasciaRecensione

Il primo metodo che abbiamo scelto di testare è il metodo **Lascia Recensione** del back-end. Questo metodo accetta due parametri: il codice fiscale dell'agente da recensire e la valutazione assegnata.

Per il codice fiscale dell'agente, è stato adottato il seguente pattern:

$^{\wedge} [A-Z] \{6\} [0-9] \{2\} [A-Z] [0-9] \{2\} [A-Z] [0-9] \{3\} [A-Z] \$$

| Parametro      | Classe di Equivalenza                    | Validità   |
|----------------|--|------------|
| Codice Fiscale | CE1: Null                                | Non valida |
|                | CE2: Stringa vuota                       | Non valida |
|                | CE3: Stringa che non rispetta il pattern | Non valida |
|                | CE4: Stringa che rispetta il pattern     | Valida     |
| Valutazione    | CE1: $\leq 0$                            | Non valida |
|                | CE2: $> 5$                               | Non valida |
|                | CE3: $1 \leq \text{Valutazione} \leq 5$  | Valida     |

Table 1: Classi di Equivalenza e validità dei parametri per il metodo lascia recensione

Adottiamo il criterio di copertura **R-WECT** (Weak Robust Equivalence Class Testing), che offre un buon livello di copertura riducendo, al contempo, il numero di test case necessari. Questo approccio rappresenta un compromesso efficace tra i criteri SECT e WECT.

Le classi di equivalenza non valide vengono testate separatamente da quelle valide, mentre ciascuna classe valida viene verificata una sola volta. In totale, il numero di test case risultante è pari a sei.

```

class ClienteControllerTest {

    private ClienteController controller;

    @BeforeEach
    void setUp() {
        controller = new ClienteController();
    }

    private LasciaRecensioneRequestDTO creaRecensione(String agenteDaRecensire, int valutazione) {
        LasciaRecensioneRequestDTO recensioneDTO = new LasciaRecensioneRequestDTO();
        recensioneDTO.setAgenteDaRecensire(agenteDaRecensire);
        recensioneDTO.setValutazione(valutazione);
        return recensioneDTO;
    }
}

```

Figure 2: Codice JUnit per il metodo Lascia recensione

```

@Test
void testLasciaRecensione_CodiceFiscaleNull_ValutazioneUno() {
    //Arrange
    LasciaRecensioneRequestDTO recensioneDTO = creaRecensione( agenteDaRecensire: null, valutazione: 1);

    //Act
    Response response = controller.lasciaRecensione(recensioneDTO);

    //Assert
    assertEquals(Response.Status.BAD_REQUEST.getStatusCode(), response.getStatus());
}

@Test
void testLasciaRecensione_CFEempty() {
    //Arrange
    LasciaRecensioneRequestDTO recensioneDTO = creaRecensione( agenteDaRecensire: "", valutazione: 2);
    //Act
    Response response = controller.lasciaRecensione(recensioneDTO);
    //Assert
    assertEquals(Response.Status.BAD_REQUEST.getStatusCode(), response.getStatus());
}

```

Figure 3: Codice JUnit per il metodo Lascia Recensione

<sup>1</sup>

---

<sup>1</sup> Al metodo testLasciaRecensione\_CodiceFiscaleNull\_ValutazioneUno  
è stato lasciato volontariamente un errore di Maintainability per chiarezza del lettore,  
verrà aggiustato in seguito

```

@Test
void testLasciaRecensione_CFNonValido() {
    //Arrange
    LasciaRecensioneRequestDTO recensioneDTO = creaRecensione( agenteDaRecensire: "SVN", valutazione: 5);

    //Act
    Response response = controller.lasciaRecensione(recensioneDTO);

    //Assert
    assertEquals(Response.Status.BAD_REQUEST.getStatusCode(), response.getStatus());
}

@Test
void testLasciaRecensione_ValutazioneNegativa() {
    //Arrange
    LasciaRecensioneRequestDTO recensioneDTO=creaRecensione( agenteDaRecensire: "SVNGLN03T05H892R", valutazione: -1);

    //Act
    Response response = controller.lasciaRecensione(recensioneDTO);

    //Assert
    assertEquals(Response.Status.BAD_REQUEST.getStatusCode(), response.getStatus());
}

```

Figure 4: Codice JUnit per il metodo Lascia recensione

```

@Test
void testLasciaRecensione_ValutazioneMaggioreDiCinque() {
    //Arrange
    LasciaRecensioneRequestDTO recensioneDTO = creaRecensione( agenteDaRecensire: "SVNGLN03T05H892R", valutazione: 6);
    //Act
    Response response = controller.lasciaRecensione(recensioneDTO);
    //Assert
    assertEquals(Response.Status.BAD_REQUEST.getStatusCode(), response.getStatus());
}

@Test
void testLasciaRecensione_CodiceFiscaleValido_ValutazioneValida() {
    //Arrange
    LasciaRecensioneRequestDTO recensioneDTO = creaRecensione( agenteDaRecensire: "SVNGLN03T05H175V", valutazione: 4);
    //Act
    Response response = controller.lasciaRecensione(recensioneDTO);
    //Assert
    assertEquals(Response.Status.OK.getStatusCode(), response.getStatus());
}

```

Figure 5: Codice JUnit per il metodo Lascia recensione

### 16.2.2 checkVisitaPerCliente

Il secondo metodo che abbiamo scelto di testare è il metodo **checkVisitaPerCliente** del back-end. Questo metodo accetta due parametri: Un identificativo dell'immobile e l'email del cliente e controlla che il cliente con quella email abbia già una visita per quell'immobile.

Per l'email del cliente, è stato adottato il seguente pattern:

`^ [a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$`

| Parametro   | Classe di Equivalenza   | Validità   |
|-------------|---|--|
| Email       | CE1: Null<br>CE2: Stringa vuota<br>CE3: Stringa che non rispetta il pattern<br>CE4: Stringa che rispetta il pattern | Non valida<br>Non valida<br>Non valida<br>Valida |
| Id immobile | CE1: Null<br>CE2: < 0<br>CE3: = 0<br>CE4: > 0   | Non valida<br>Non valida<br>Valida<br>Valida     |

Table 2: Classi di Equivalenza e validità dei parametri per il checkVisitaPerCliente

Adottiamo anche qui il criterio di copertura **R-WECT** (Weak Robust Equivalence Class Testing). Quindi in totale, il numero di test case risultante è pari a **sette**.

```

class VisitaControllerTest {
    private VisitaController controller;

    @BeforeEach
    void setUp() {
        controller = new VisitaController();
    }

    private Response callCheckVisita(Integer idImmobile, String email) {
        return controller.checkVisitaPerCliente(idImmobile, email);
    }

    @Test
    void testCheckVisita_EmailNull(){
        //Arrange
        int idImmobile = 0;
        //Act
        Response response = callCheckVisita(idImmobile, email: null);
        //Assert
        assertEquals(Response.Status.BAD_REQUEST.getStatusCode(), response.getStatus());
    }
}

```

Figure 6: Codice JUnit per il metodo checkVisitaPerCliente

```

@Test
void testCheckVisita_EmailValidIdZero(){
    //Arrange
    String email = "email123@gmail.com";
    int idImmobile = 0;
    //Act
    Response response = callCheckVisita(idImmobile, email);
    //Assert
    assertEquals(Response.Status.BAD_REQUEST.getStatusCode(), response.getStatus());
}

@Test
void testCheckVisita_EmailValidIdPositive(){
    //Arrange
    String email = "email123@gmail.com";
    int idImmobile = 1;
    //Act
    Response response = callCheckVisita(idImmobile, email);
    //Assert
    assertEquals(Response.Status.OK.getStatusCode(), response.getStatus());
}

```

Figure 7: Codice JUnit per il metodo checkVisitaPerCliente

```

    @Test
    void testCheckVisita_EmailValidIdNull(){
        //Arrange
        String email = "email123@gmail.com";
        //Act
        Response response = callCheckVisita( idImmobile: null, email);
        //Assert
        assertEquals(Response.Status.BAD_REQUEST.getStatusCode(), response.getStatus());
    }

    @Test
    void testCheckVisita_EmailValidIdNegativ(){
        //Arrange
        String email = "email123@gmail.com";
        int idImmobile = -1;
        //Act
        Response response = callCheckVisita(idImmobile, email);
        //Assert
        assertEquals(Response.Status.BAD_REQUEST.getStatusCode(), response.getStatus());
    }
}

```

Figure 8: Codice JUnit per il metodo checkVisitaPerCliente

```

    @Test
    void testCheckVisita_EmailEmpty(){
        //Arrange
        String email = "";
        int idImmobile = 1;
        //Act
        Response response = callCheckVisita(idImmobile, email);
        //Assert
        assertEquals(Response.Status.BAD_REQUEST.getStatusCode(), response.getStatus());
    }

    @Test
    void testCheckVisita_EmailInvalid(){
        //Arrange
        String email = "invalidEmail";
        int idImmobile = 3;
        //Act
        Response response = callCheckVisita(idImmobile, email);
        //Assert
        assertEquals(Response.Status.BAD_REQUEST.getStatusCode(), response.getStatus());
    }
}

```

Figure 9: Codice JUnit per il metodo checkVisitaPerCliente

---

<sup>2</sup>Al metodo testCheckVisita\_EmailEmpty  
è stato lasciato volontariamente un errore di Maintainability per chiarezza del lettore,  
verrà aggiustato in seguito

## 17 Usabilità sul campo

Per la valutazione dell'usabilità di DietiEstates25, abbiamo adottato diverse tecniche. Una delle prime tecniche utilizzate durante la fase di progettazione del sistema è stata la raccolta di feedback dagli utenti finali dell'applicazione.

In particolare, nella fase iniziale del design, avevamo alcuni dubbi sulla struttura delle pagine di registrazione e creazione degli account. Per questo motivo, abbiamo deciso di sottoporre le diverse proposte agli utenti finali, chiedendo loro un parere.

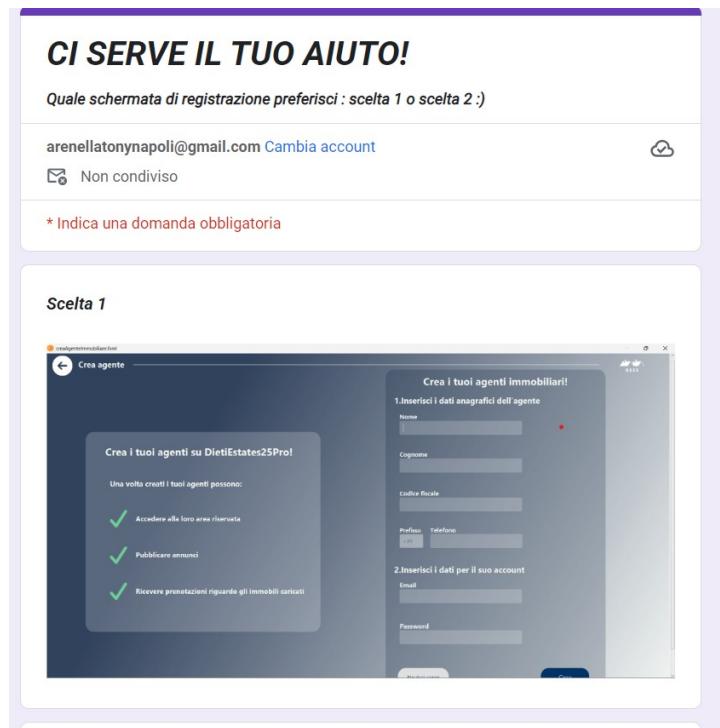


Figure 10

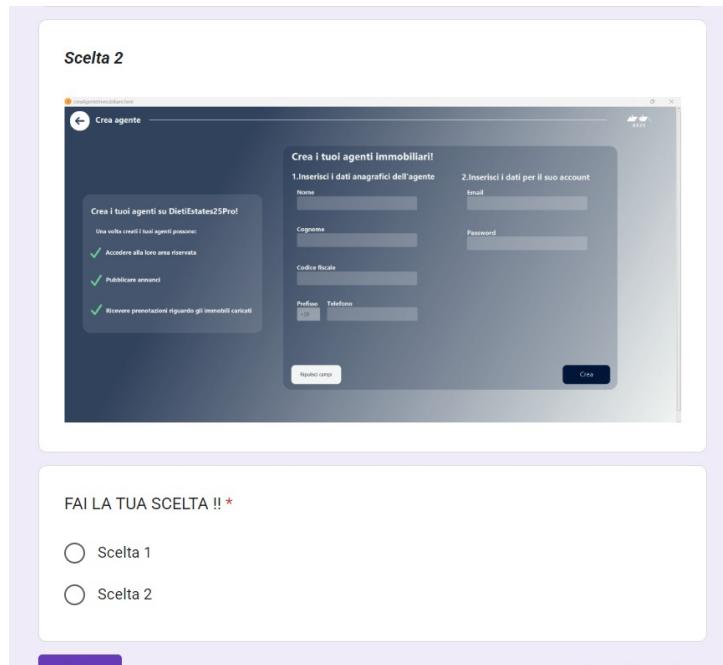


Figure 11

Abbiamo quindi condotto un primo sondaggio, chiedendo agli utenti quale delle schermate preferiscono.

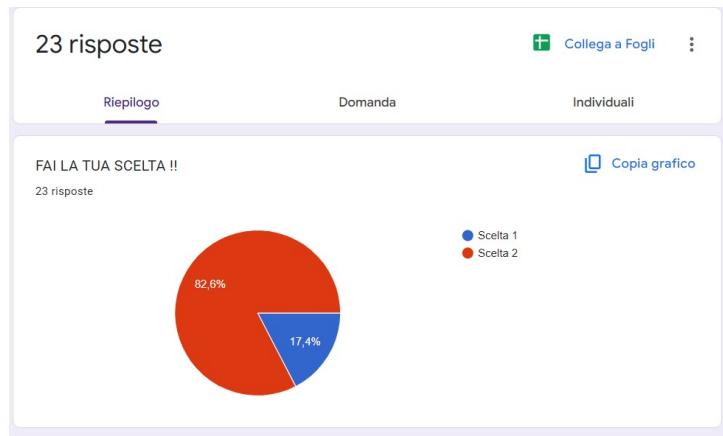


Figure 12

La seconda schermata è risultata la più apprezzata. Approfondendo le motivazioni della scelta attraverso interviste agli utenti, è emerso che questa versione è stata ritenuta più ordinata e leggibile, grazie a una disposizione degli elementi più chiara e intuitiva.

Successivamente abbiamo adottato questo design anche per la pagina di registrazione al sistema. Un ulteriore sondaggio.

Abbiamo fatto anche un secondo sondaggio agli utenti finali, in questo caso riguardava il design del processo di lascia recensione. Agli utenti è stato chiesto quale flusso preferiscono:



Figure 13



Caso 1: apertura del profilo dell'agente con la parte di valutazione e recensione tutta a destra

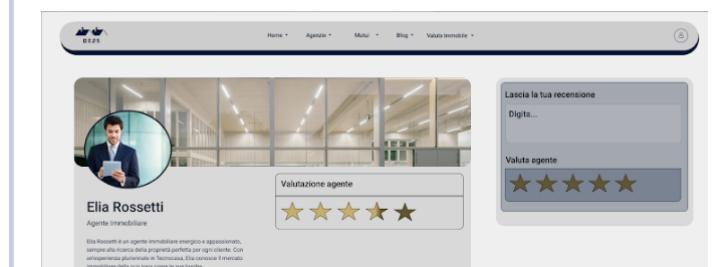
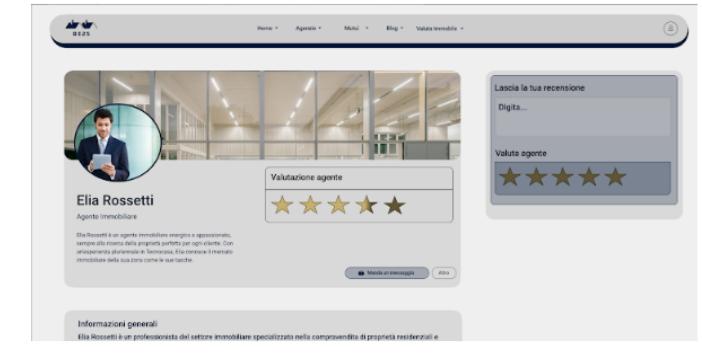


Figure 14

Caso 2: apertura del profilo dell'agente con la valutazione vicino al nome ma con il lascia recensione a destra



Caso 3: pop up che esce sulla pagina appena si clicca il pulsante che ti permette di lasciare la recensione e valutare l'agente

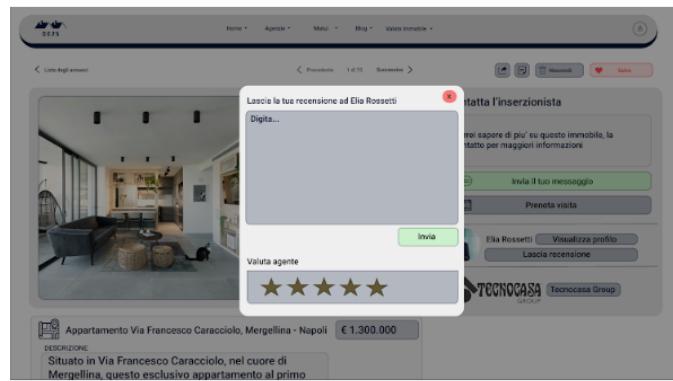


Figure 15

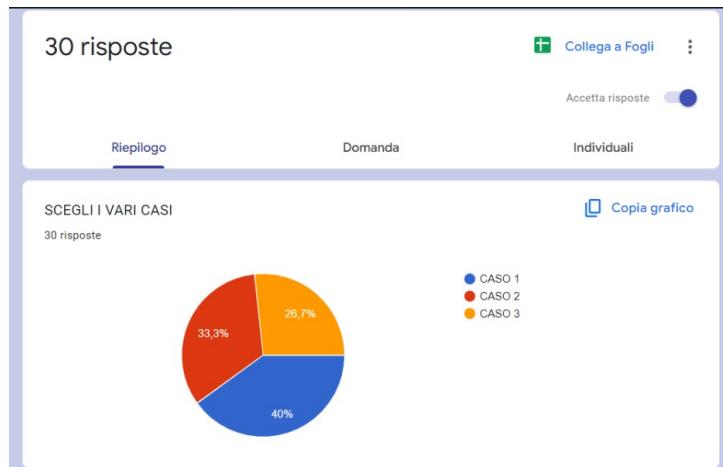


Figure 16

Il processo vincitore è stato il terzo(come ci aspettavamo), quindi il design del processo di lascia recensione si è basato su questo feedback degli utenti finali.

## 18 Valutazione degli esperti

In questa parte ci concreteremo sulla valutazione dell'usabilità di DietiEstates25 da parte di esperti, un metodo in cui esperti di usabilità o di applicazioni specifiche del dominio valutano l'interfaccia utente per identificare problemi, valutare la qualità del design e fornire feedback concreti.

Una prima tecnica che abbiamo usato per usare la valutazione degli esperti è stata la valutazione euristica basta sulle **otto regole d'oro di Shneiderman**

1. **Sforzarsi per la coerenza** Utilizzare terminologie, layout, colori e comportamenti coerenti in tutta l'interfaccia.
2. **Abilitare scorciatoie per gli utenti esperti** Offrire scorciatoie da tastiera, macro e menu personalizzabili per velocizzare le operazioni.
3. **Fornire un feedback informativo** Ogni azione dell'utente deve generare un feedback chiaro e immediato sullo stato del sistema.
4. **Progettare dialoghi per chiudere un'azione** Strutturare i flussi di lavoro in fasi logiche con chiusure chiare, come la conferma dopo l'invio di un modulo.
5. **Prevenire gli errori** Progettare l'interfaccia per minimizzare gli errori, offrendo conferme, validazioni e annullamenti delle operazioni.
6. **Consentire annullamenti semplici delle azioni** Gli utenti dovrebbero poter annullare o ripetere operazioni senza conseguenze gravi.
7. **Favorire il controllo da parte dell'utente** L'utente deve sentirsi al comando e non costretto da interazioni rigide o automatizzate.
8. **Ridurre il carico cognitivo** Minimizzare la complessità e il sovraccarico di informazioni, visualizzando solo ciò che è necessario al momento giusto.

Facendo un'analisi approfondita della nostre interfaccie utente, siamo giunti alla seguente conclusione:

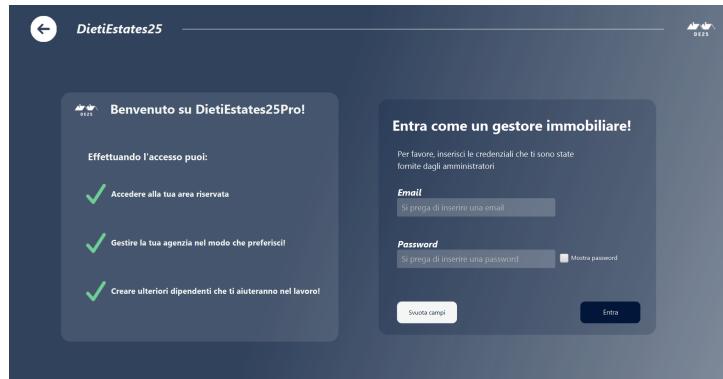


Figure 17

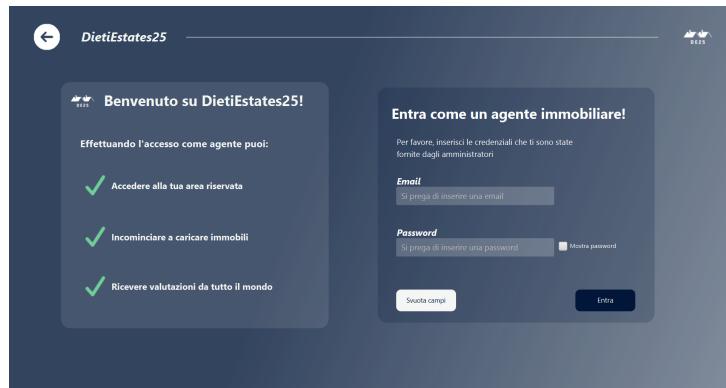


Figure 18

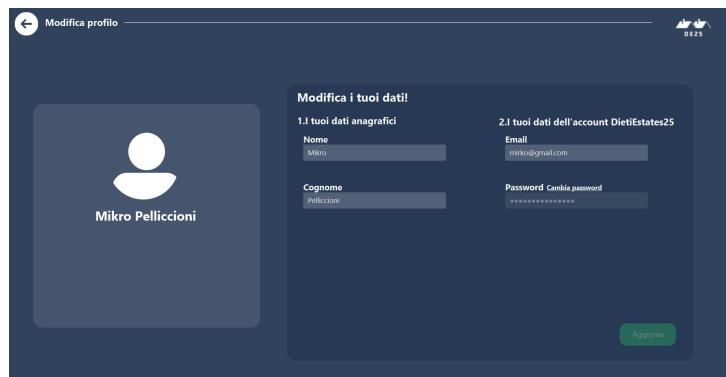


Figure 19

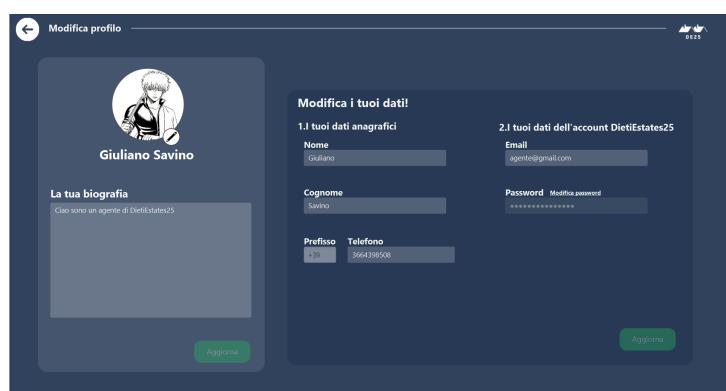


Figure 20

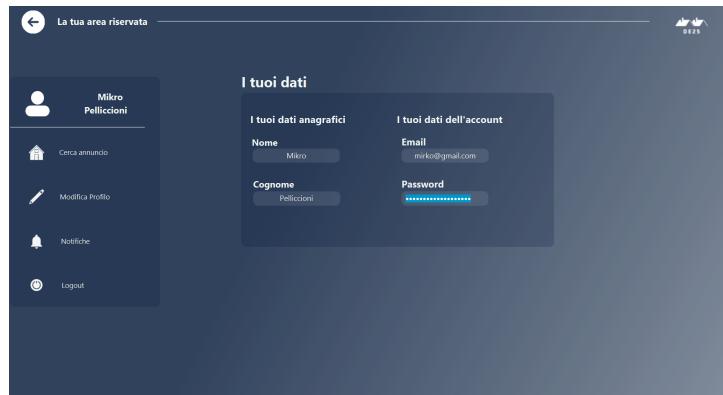


Figure 21

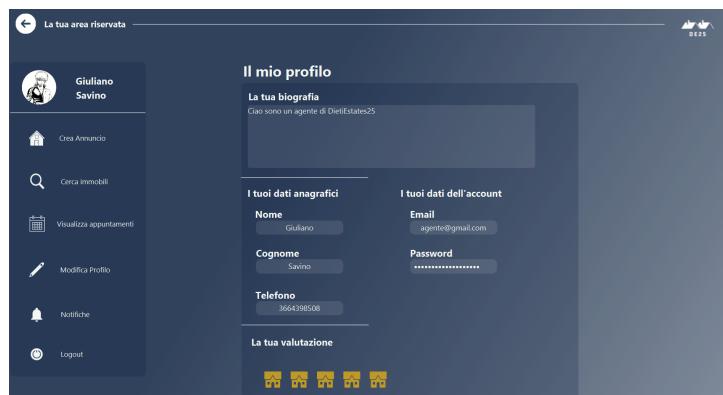


Figure 22

Dall'analisi di queste interfacce utente, riteniamo che il criterio di **coerenza** sia stato rispettato. In particolare:

- I vantaggi dell'accesso sono sempre elencati sulla sinistra, mentre le informazioni principali si trovano sulla destra della pagina.
- L'area riservata presenta un layout e comportamenti coerenti per tutti i tipi di utenti che utilizzano il sistema.
- La sezione di modifica del profilo è uniforme per tutti gli utenti del sistema, con eventuali variazioni dovute a specifiche funzionalità richieste dal committente.

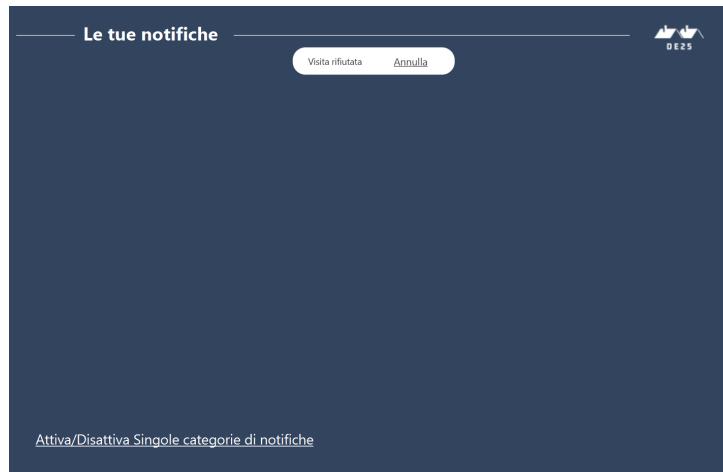


Figure 23

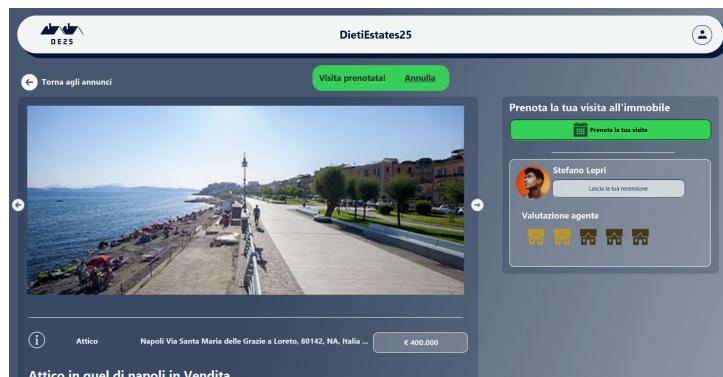


Figure 24

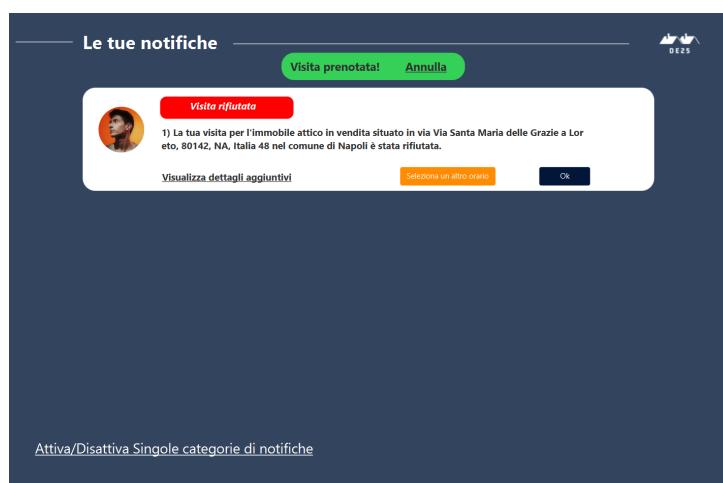


Figure 25

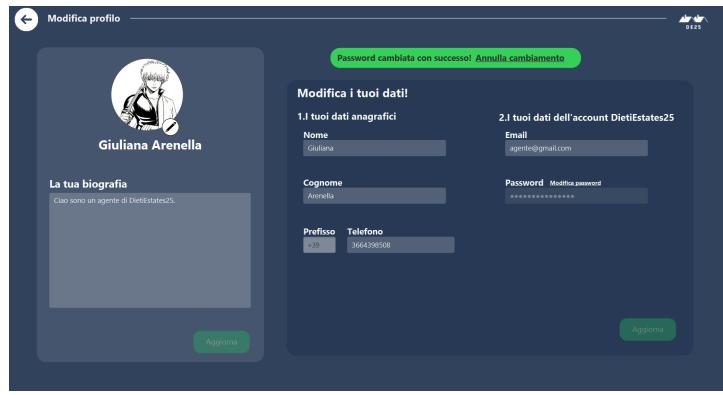


Figure 26

Ogni principale azione dell’utente genera un feedback chiaro e immediato sulla stato del sistema. **Facendo dei test di usabilità** nelle fasi finali del sistema ci siamo resi conto che nella sezione modifica profilo e nel lascito della recensione non davamo feedback all’utente quindi abbiamo deciso di aggiungerli.

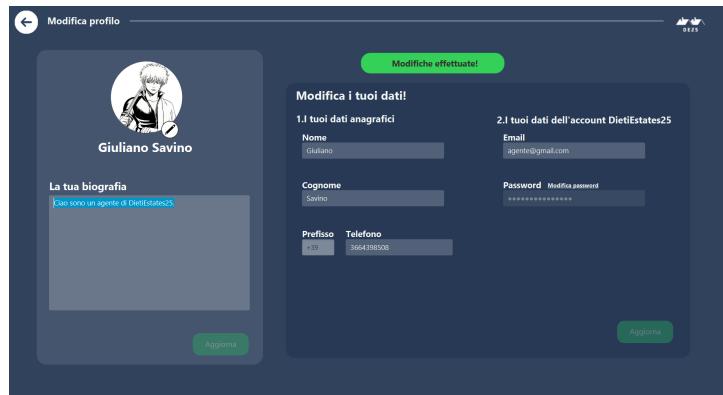


Figure 27

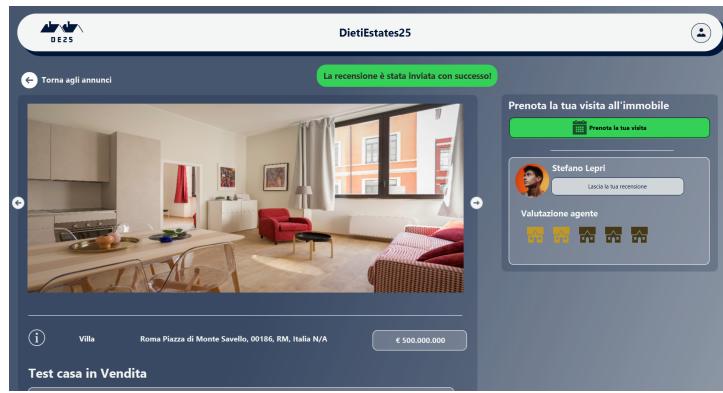


Figure 28

## 18.1 Test con utenti reali

Abbiamo selezionato un gruppo rappresentativo di utenti, utenti che hanno familiarità con la tecnologia, utenti anziani che hanno difficoltà visive e agenti immobiliari. Le attività testate includono:

- Registrazione account

- Prenotare visita di un immobile
- Lasciare una recensione
- Caricare immobile
- Modificare il proprio profilo
- Creazione agente
- Annullamento operazioni

Sono stati effettuati questi test con un sistema in uno stato finale.

In particolare, nelle tabelle seguenti sono riportati le funzionalità testate e per ogni utente viene indicato :

- S= Successo(ovvero la task è stata completata velocemente senza intoppi)
- SP = Successo parziale(ovvero è stato necessario un po' di tempo in più e piccoli suggerimenti)
- F= Fallimento(ovvero la task non è stata completata nonostante i piccoli suggerimenti)

## 19 Risoluzione errori

Dalle valutazioni effettuate, sono state riscontrate alcune criticità che hanno comportato una rielaborazione di alcuni aspetti dell'interfaccia.

| Segnalazione riscontrata   | Risoluzione  |
|--|--|
| L'utente, quando caricava un immobile, voleva un messaggio di conferma prima di procedere con il caricamento.  | È stato aggiunto un messaggio di conferma prima di caricare un immobile.             |
| L'utente, quando effettuava il logout, voleva un messaggio di conferma prima di uscire.  | È stato aggiunto un messaggio di conferma prima del logout.                          |
| Abbiamo riscontrato una problematica nei feedback della pagina di modifica profilo: avevamo dato per scontato che modificando i dati nei campi di testo, l'utente si accorgesse immediatamente del cambiamento. Tuttavia, per alcuni utenti questo non era chiaro. | Abbiamo aggiunto feedback positivi esplicativi nella pagina di modifica del profilo. |
| Alcuni utenti hanno segnalato difficoltà dovute alla mancanza di una funzione di autocompletamento.  | È stato aggiunto l'autocompletamento nei campi appropriati.                          |

Table 3: Segnalazioni riscontrate e relative risoluzioni

| Nome    | Registrazione+Login | Prenota visita + lascia recensione | Carica immobile | Accetta richiesta di visita | Percentuale superamento |
|---------|---------------------|------------------------------------|-----------------|-----------------------------|-------------------------|
| Antonio | ✓ (100)             | ✓ (100)                            | 50              | ✓ (100)                     | 50                      |
| Matteo  | ✓ (100)             | ✓ (100)                            | ✓ (100)         | ✓ (100)                     | 50                      |
| Luca    | ✓ (100)             | ✓ (100)                            | ✓ (100)         | ✓ (100)                     | 50                      |
| Fortuna | ✓ (100)             | ✓ (100)                            | ✓ (100)         | ✓ (100)                     | 50                      |
| Adele   | 50                  | ✓ (100)                            | 50              | ✓ (100)                     | 50                      |

Table 4: Tabella dei punteggi e delle percentuali di superamento