| **Università degli Studi di Salerno Corso di Ingegneria del Software** |
| --- |

**Avenue 814**

**System Design Document  
Versione 2.1**

****

Data: 25/11/2021

**Coordinatore del progetto:**

| **Nome** | **Matricola** |
| --- | --- |
|  |  |
|  |  |

**Partecipanti:**

| **Nome** | **Matricola** |
| --- | --- |
| Pisu Antonio | 0512108616 |
| Renzulli Marta | 0512107377 |
| Ricco Maurizio | 0512108622 |
|  |  |
|  |  |
|  |  |

| **Scritto da:** | PISU A.,RENZULLI M.,RICCO M. |
| --- | --- |

**Revision History**

| **Data** | **Versione** | **Descrizione** | **Autore** |
| --- | --- | --- | --- |
| 18/11/2021 | 1.0 | Prima stesura | Renzulli Marta |
| 19/11/2021 | 1.1 | Continuo Prima stesura | Pisu Antonio,Ricco Maurizio |
| 22/11/2021 | 2.0 | Seconda stesura | Renzulli Marta,Pisu Antonio,Ricco Maurizio |
| 22/11/2021 | 2.1 | Revisione e continuo Seconda stesura | Renzulli Marta,Pisu Antonio,Ricco Maurizio |

**Indice**

1. INTRODUZIONE 5

1.1. Design Goals

1.2. Trade-off

1.3. Definizioni

2. ARCHITETTURA DEL SISTEMA CONCORRENTE 8

3. ARCHITETTURA DEL SISTEMA PROPOSTO 9

3.1. Panoramica

3.2. Decomposizione in Sottosistemi

4. SERVIZI DEI SOTTOSISTEMI 17

1. **INTRODUZIONE**

## 1.Design Goals e Trade-off

### 1.1 Design Goals

I design goal identificati per il sistema web sono i seguenti:

· Criteri di performance

o Tempo di risposta

-Per l’accesso all’area utente il tempo di risposta è di 2 secondi (RNF P1).

-Per la visualizzazione dei form di richiesta il tempo di risposta è di 1 -secondo (RNF P1).

-Per la visualizzazione delle liste dei prodotti il tempo di risposta è di 3 secondi (RNF P1).

-Per la modifica di prodotti, il tempo di risposta è di 1 secondo (RNF P1).

o Memoria

-Il sistema dovrà mantenere i dati consistenti (RNF A2).

-La dimensione complessiva del sistema dipende dalla memoria utilizzata per il mantenimento del database (Dominio applicativo).

· Criteri di affidabilità

o Robustezza

-Eventuali input non validi immessi dall’utente saranno opportunamente segnalati attraverso messaggi di errore (RNF A1).

o Affidabilità

-Il sistema deve garantire l’affidabilità dei servizi proposti. Il prodotto software sarà sviluppato in modo tale da controllare il corretto funzionamento tramite diverse tipologie di testing, black-box e white-box (Documenti di management).

o Disponibilità

-Una volta realizzato il sistema, sarà disponibile ogni qualvolta l’utente e l’amministratore ne richiederanno l’utilizzo (RNF U4).

o Security

-L’accesso al sistema è garantito mediante una email e una password (che verrà criptata) (RNF A4). Inoltre, la sicurezza è garantita in quanto verranno implementati filtri di autenticazione per accedere alle aree riservate (RIF A3).

· Criteri di costo

o Costi di sviluppo

-È stimato un costo complessivo di 200 ore per la progettazione e lo sviluppo del sistema (50 ore per ogni team member) (Documenti di management).

· Criteri di manutenzione

o Estensibilità

-Il sistema sarà facilmente estendibile in quanto sarà possibile aggiungere nuove funzionalità ad esso senza andare a modificare l’intera struttura, rispettando i pattern architetturali adottati (RNF S2).

o Leggibilità

- In concordanza con gli obiettivi di estendibilità e modificabilità il codice del sistema sarà prodotto per avere un alto grado di leggibilità (RNF S3).

o Adattabilità

-Il sistema può funzionare in ambito internazionale (Dominio applicativo).

o Tracciabilità dei requisiti

- La tracciabilità è garantita dalla fase di progettazione fino al testing (Documenti di management).

o Portabilità

-Il sistema sarà portabile in quanto l’interazione avviene mediante un browser senza interazione con il sistema sottostante, c’è quindi indipendenza dal sistema operativo (RNF I1)**.**

· Criteri utenti finali

o Usabilità

-Il sistema sarà di facile comprensione e utilizzo, sarà infatti molto semplice da apprendere senza la consultazione di documentazione associata (RNF U1, RNP U3). L’intuitività è garantita in quanto il sistema avrà una buona prevedibilità, cioè la risposta del sistema ad un’azione utente sarà corrispondente alle aspettative (RNF U2, RNF IN 1, RNF IN2).

o Utilità

-Il lavoro dell’utente verrà supportato nel miglior modo possibile dal sistema, infatti l’utente compirà le operazioni richieste senza il carico di lavoro che deriva dal realizzarle diversamente (Dominio applicativo).

### 1.2 Trade-off

1.1.1 *Comprensibilità vs costi*

Si preferisce aggiungere costi per la documentazione al fine di rendere il codice comprensibile anche alle persone non coinvolte nel progetto o le persone coinvolte che non hanno lavorato a quella sezione di progetto. Commenti diffusi nel codice facilitano la comprensione, di conseguenza migliorare la comprensibilità agevola il mantenimento e anche il processo di modifica.

1.1.2 *Tempo di risposta e Affidabilità*

Il sistema sarà implementato in modo tale da preferire l’affidabilità al tempo di risposta,per garantire un controllo più accurato dei dati in input a discapito del tempo di risposta del sistema.

1.1.3 *Manutenibilità vs efficienza*

Si preferisce avere una maggiore manutenibilità del sistema facendo in modo che ogni sottosistema non acceda direttamente al DB, ma che l’accesso ai dati venga gestito da un sottosistema intermedio a discapito dell’efficienza e delle prestazioni generali.

## 1.3 Definizioni

· **Deployment Diagram**: Diagramma UML di specifica per le relazioni tra le componenti realizzate (con relative tecnologie implementative) e le risorse Hardware e Software necessarie al corretto funzionamento del sistema;

· **Design Goal**: Obiettivi di design progettati per il sistema proposto;

· **Design Trade-off**: Scelte e compromessi tra design goals dissonanti;

· **Greenfield Engineering:** Tipologia di sviluppo che comincia da zero, non esiste nessun sistema a priori e i requisiti sono ottenuti dall’utente finale e dal cliente. Nasce, perciò, a partire dai bisogni dell’utente.

· **Pattern MVC**: Modello architetturale del sistema a tre livelli;

· **Model**: Layer del pattern architetturale per la gestione e memorizzazione dei dati persistenti;

· **View**: Layer del pattern architetturale per la gestione e controllo d’interfaccia tra le risorse del sistema e l’utente finale;

· **Controller**: Layer del pattern architetturale per la gestione e il controller della logica di business.

2. Architettura del Sistema corrente

Attualmente non esiste un sistema software che si occupa di gestire questa problematica, ossia la gestione degli ordini. Gli ordini vengono gestiti tramite scambi di email tra l'utente e il sito web, ciò risulta essere molto macchinoso e alquanto lento poiché i tempi necessari allo scambio di email e il recupero dei dati da parte dell’ utente non sono agevoli. Per quanto riguarda la disponibilità dei prodotti, viene compilato manualmente dall’amministratore, per poi essere revisionato e convalidato periodicamente. Quindi si tratta di un sistema che rientra nel campo della Greenfield Engineering.

3. Architettura del Sistema Proposto

3.1 Panoramica

Il sistema da noi proposto è un’applicazione Web che permette agli utenti registrati e non registrati di poter visualizzare prodotti di abbigliamento di qualsiasi tipo a scelta dell’utente. L’obiettivo che si pone è quello di fornire uno strumento di gestione sia dei pagamenti che l’utente registrato ha intenzione di acquistare prodotti scelti nel proprio carrello, sia per la compilazione in fase di registrazione/login e inserimento prodotti(tale operazione verrà offerta solo all’amministratore), che per la ricerca veloce dei prodotti. Detto in breve, il sistema metterà a disposizione interfacce all’utente per poter effettuare una registrazione, l’accesso nel momento in cui si ha già un account, per visualizzare il proprio carrello e di conseguenza poter compiere l’acquisto, per poter modificare i propri dati personali, e infine da parte dell’amministratore per la gestione dei prodotti.

Verrà utilizzata l’architettura **MVC**(Model View Control), dove il model metterà a disposizione le operazioni per accedere ai dati utili all’applicazione, ed implementerà quindi la struttura dati centrale(database), invece, il controller gestirà il controllo del flusso, ovvero ottiene gli input dall’utente tramite la view e manda le informazioni al model. La view visualizzerà il model e verrà notificato ogni volta che il model sarà modificato. I gestori saranno individuati in base alle funzionalità per poter rendere massima la coesione e minimo l’accoppiamento tra i sottosistemi in modo che i cambiamenti in un sottosistema non influiscano sugli altri.

3.2 Decomposizione in Sottosistemi

3.2.1 Decomposizione in Layer

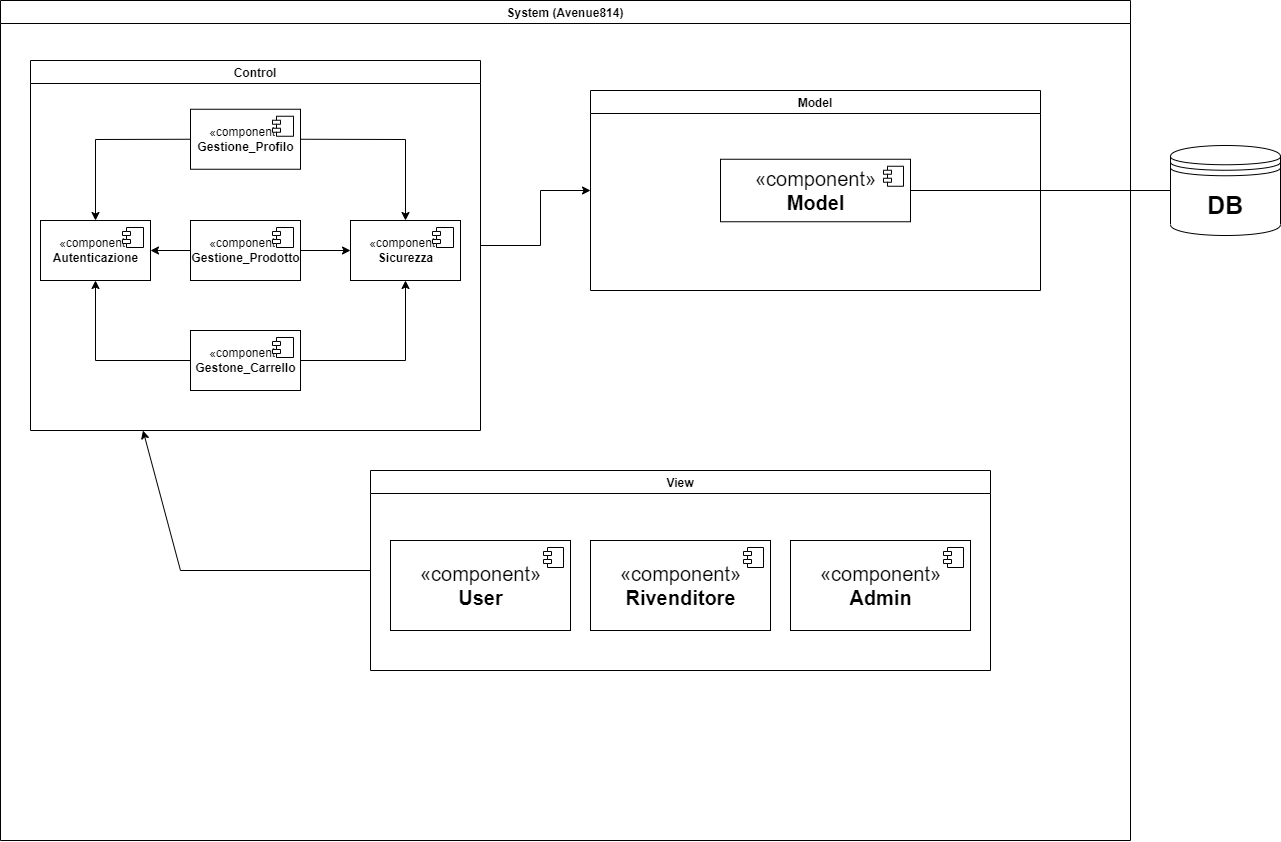
Nell’architettura MVC, i sottosistemi sono classificati in tre differenti tipi:

* **Model**: si occupa della gestione e dello scambio dei dati tra i sottosistemi;
* **View**: raccoglie e gestisce elementi di interfaccia grafica e gli eventi generati su di essi;
* **Control**: si occupa della gestione della logica del sistema;



3.2.2 Decomposizione in Sottosistemi

Dopo un’analisi funzionale dettagliata, si è deciso di gestire i singoli componenti con basso accoppiamento ed elevata coesione in modo tale da garantire, in caso di successive modifiche, il minor numero di aggiornamenti da apportare tra tutti i sottosistemi. Abbiamo deciso di distinguere le funzionalità per area di gestione (User, Client e Amministrator) e creato un’interfaccia intermedia tra i sistemi di logica di business e il database, offrendo allo storage un’interfaccia più stabile rispetto al database e quindi nel caso in cui cambi l’interfaccia del sottosistema database, solo il sottosistema storage verrà modificato.



3.2.3 Hardware/Software Mapping

Il sistema che si desidera sviluppare utilizzerà una struttura hardware costituita da un Server che risponderà ai server richiesti dai client. Il Client è una qualsiasi macchina attraverso il quale un utente può collegarsi, utilizzando una connessione Internet, per accedere al sistema, mentre il Server è una macchina, scelta dal team di sviluppatori, per gestire la logica di business e i dati persistenti contenuti nel database. Il client e il server saranno connessi tramite il protocollo HTTP, dove il primo manda delle richieste e il secondo le risponde. Le componenti hardware e software necessarie per il client sono un Web Browser(es. Google Chrome, Firefox, ecc…) e infine una connessione Internet in modo tale da poter accedere al servizio.

In conclusione, per il server, si ha la necessità di avere una macchina in grado di interfacciarsi ad internet e di aver la capacità di memorizzare dei dati che per il sistema ritiene fondamentali. Detto in breve, avere un sistema di DBMS.

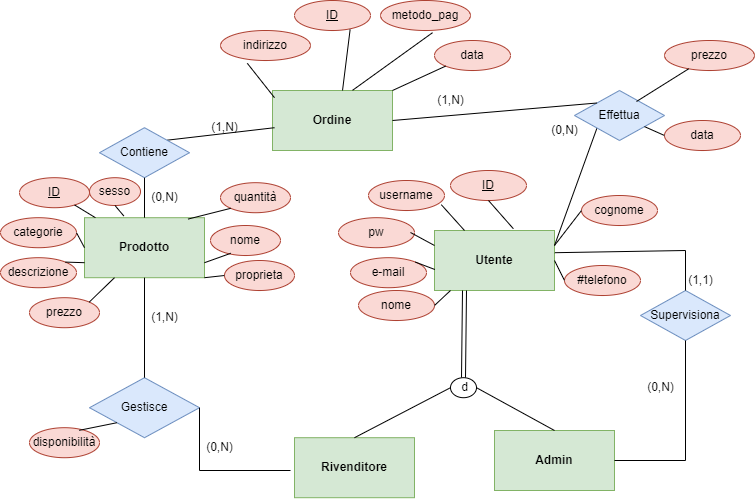
3.2.4 Identificazione e memorizzazione dei dati persistenti

Per la memorizzazione dei dati si è scelto un Database relazionale che consente un accesso efficiente ai dati, in termini di risposta e un ampio spazio di archiviazione. Per questioni di efficienza e maggior usabilità si è deciso di rendere persistenti le informazioni riguardo al cliente che si autentica al sistema, ossia al USER, in modo tale che il cliente quando dovrà effettuare un secondo accesso, esso viene riconosciuto, senza problemi, dal sistema.

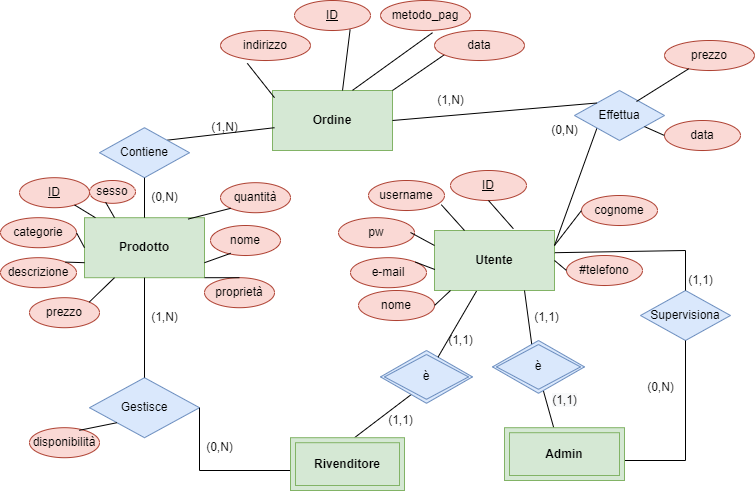
Si ha la necessità di aggiungere ulteriori informazioni nel database, ovvero i PRODOTTI, che permettono con estrema facilità l’aggiornamento automatico della vetrina. Si ha il bisogno, però, di affiancare il database con un file esterno(ovvero una semplice cartella) dove al suo interno sono contenute delle immagini, ognuna associata al proprio prodotto, che verranno mostrate nella vetrina. La motivazione di tale scelta è che l'utilizzo delle immagini non è fortemente necessario e quindi si è optato di trovare una sistema semplice e poco complesso nel gestirla, in un secondo momento verrà implementata all’interno del DBMS.

Infine, altre informazioni necessarie per il maggior funzionamento del sistema è quello di gestire gli ORDINI, cioè le informazioni necessarie per il cliente che ha effettuato un acquisto, dove quest’ultimo ha la possibilità di controllare la propria cronologia di acquisti.

**SCHEMA ER**



**SCHEMA ER RISTRUTTURATO**

****

**MAPPING**

**Ordine**(ID, indirizzo, metodo\_pag,data).

**Contiene**(Ordine.ID ↑,Prodotto.ID ↑).

**Prodotto**(ID,sesso,proprieta,disponibilita,categoria,quantità,nome,descrizione,prezzo).

**Utente**(ID,nome,cognome,username,pw,e-mail,#telefono,ruolo).

**Effettua**(data,prezzo,Utente.ID ↑,Ordine.ID ↑).

**Rivenditore**(Utente.ID ↑).

**Gestisce**(disponibilità,Rivenditore.Utente.ID ↑,Prodotto.ID ↑).

**Admin**(Utente.ID ↑)

**Ordine**

| **Nome** | **Tipo** | **Null** | **Key** |
| --- | --- | --- | --- |
| ID | int | Not null | Primary key |
| indirizzo | Varchar(30) | Not null |  |
| metodo\_pag | Varchar(25) | Not null |  |
| data | Varchar(45) | Not null |  |

**Prodotto**

| **Nome** | **Tipo** | **Null** | **Key** |
| --- | --- | --- | --- |
| ID | int | Not null | Primary key |
| categoria | Varchar(150) |  |  |
| quantità | int |  |  |
| nome | Varchar(25) |  |  |
| proprietà | int |  |  |
| descrizione | Varchar(80) |  |  |
| prezzo | Double |  |  |
| sesso | char(15) |  |  |
| disponibilità | boolean | default true | Foreign key |

**Utente**

| **Nome** | **Tipo** | **Null** | **Key** |
| --- | --- | --- | --- |
| ID | int | Not null | Primary key |
| nome | Varchar(16) |  |  |
| cognome | Varchar(16) |  |  |
| username | Varchar(16) |  |  |
| pw | Varchar(45) | Not null |  |
| e-mail | Varchar(30) | Not null |  |
| #telefono | int |  |  |
| ruolo | Varchar(25) |  |  |

**Contiene**

| **Nome** | **Tipo** | **Null** | **Key** |
| --- | --- | --- | --- |
| ID\_ordine | int | Not null | Primary key/Foreign key |
| ID\_prodotto | int | Not null | Primary key/Foreign key |

**Effettua**

| **Nome** | **Tipo** | **Null** | **Key** |
| --- | --- | --- | --- |
| ID\_ordine | int | Not null | Primary key/Foreign key |
| ID\_utente | int | Not null | Primary key/Foreign key |
| data | Date | Not null |  |
| prezzo | Double |  |  |

**Rivenditore**

| **Nome** | **Tipo** | **Null** | **Key** |
| --- | --- | --- | --- |
| ID\_utente | int | Not null | Primary key/Foreign key |

**Gestisce**

| **Nome** | **Tipo** | **Null** | **Key** |
| --- | --- | --- | --- |
| ID\_prodotto | int | Not null | Primary key/Foreign key |
| ID\_rivenditore | int | Not null | Primary key/Foreign key |
| disponibilità | boolean | default true |  |

**Admin**

| **Nome** | **Tipo** | **Null** | **Key** |
| --- | --- | --- | --- |
| ID\_utente | int | Not null | Primary key/Foreign key |

3.2.3 Controllo degli accessi e sicurezza

Nel sistema, come espresso precedentemente, sono presenti 3 attori che interagiscono con esso, ognuno dei quali ha almeno una funzionalità diversa dagli altri attori. Per assicurare che tali attori non vengano riconosciuti in maniera errata oppure vengono considerati come lo stesso attore, viene fatto, nel momento in cui si mandi una richiesta al sistema, un controllo in modo tale da riconoscere quel determinato ruolo e permettergli l’utilizzo di un determinato servizio. Detto in breve, consideriamo l’accesso al sistema da parte del Cliente, ovvero colui che può visualizzare prodotti e successivamente effettuare un acquisto, quest’ultimo può svolgere solo queste operazioni. Nel caso in cui, utilizzando metodi di accesso non comuni, dovesse accedere a servizi(pagine JSP) che in fase di progettazione non dovrebbe accedere, il sistema controllerà quell’entità e farà la verifica se quest’ultima può accedere o meno. Tale operazione viene fatta analogamente agli altri attori e servizi. Al livello di sicurezza, si è implementato una strategia di codifica/decodifica dei dati più sensibili, in questo caso le password degli utenti. Tale strategia è il BASE64, tale scopo è quello di trasformare una sequenza di dati binari in stringhe di testo ASCII, detto in breve all’interno del DB le password verranno salvate codificate e successivamente decodificate in momenti di utilizzo.

3.2.4 Controllo del flusso globale del sistema

Il sistema *Avanue814* fornisce funzionalità che richiedono una continua interazione da parte dell’utente, per tale ragione abbiamo adottato un controllo del flusso globale del sistema di tipo event-driven, che è un tipo di controllo flessibile e guidato dagli eventi.

3.2.5 Identificazione condizioni limite

**Start-up:** Per il primo avvio del sistema “*Avenue814”* è necessario l’avvio di un web Server che fornisca il servizio di un Database per la gestione dei dati persistenti e l’interpretazione ed esecuzione del codice lato server. Successivamente, tramite l’interfaccia di Login, sarà possibile autenticarsi tramite opportune credenziali. Una volta effettuato l’accesso, il sistema porterà l’utente alla Homepage, dalla quale potrà usufruire i servizi che gli vengono offerti.

**Terminazione:** Al momento della chiusura dell’applicazione si ha la terminazione del sistema con un regolare Logout dal sistema. Invece per ottenere una corretta terminazione dell’intero sistema, l’amministratore dovrà effettuare la procedura di terminazione, dopo la quale nessun client potrà connettersi.

**Gestione delle eccezioni:** Nel sistema possono verificarsi diversi casi di fallimento:

* Un caso di fallimento comune, è l’interruzione inaspettata del sistema, dove a riguardo non sono previsti metodi di recupero della sessione interrotta, escludendo il mantenimento dei dati tramite backup.
* Un altro caso di fallimento potrebbe essere dovuto ad un errore critico nell’hardware, non è prevista alcuna misura correttiva.
* Un altro caso di fallimento potrebbe essere dovuto ad un mancato salvataggio di dati causato da un malfunzionamento del sistema.

# 4. Servizi dei Sottosistemi

**View**: Interfacce che gestiscono l’interfaccia grafica e gli eventi generati dall’interazione dell’utente con il sistema. Suddivisa in:

· **Amministratore**

· **Utente**

· **Rivenditore**

**Gestione\_Carrello** offre 3 servizi all’interfaccia Utente:

* **Visualizzazione stato carrello**
* **Aggiunta/Rimozione prodotti**

**Autenticazione** offre 3 servizi all’interfaccia Utente:

* **Verifica dell’identità**
* **Login**
* **Logout**

**Gestione\_profilo** offre 2 servizi all’interfaccia Utente:

* **Modifica dati personali**
* **Visualizzazione profilo**