UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI INGEGNERIA
Dipartimento di Ingegneria dell'Informazione
Corso di Laurea in Ingegneria Informatica e dell'Automazione

TESI DI LAUREA

Progettazione e implementazione di un sistema per la gestione di una clinica privata

Design and implementation of a system for managing a private clinic

Relatore

Prof. Domenico Ursino

Candidato

Emanuele Frisi

ANNO ACCADEMICO 2022-2023

Anche davanti al suo creatore
la bestia non conosce gratitudine

Spike Spiegel

Sommario

Nel vasto panorama dell'ingegneria del software, uno dei task più stimolanti e complessi è la progettazione e lo sviluppo di software su misura, per soddisfare le esigenze specifiche di un'organizzazione. Nella presente tesi ci troviamo di fronte a una sfida insolita: creare un software dedicato alla gestione di una clinica privata immaginaria di nome "Casa Alfredo". Questo progetto non è semplicemente un esercizio accademico, ma un'opportunità per scoprire come le tecnologie informatiche possano migliorare l'efficienza operativa e l'esperienza dei pazienti all'interno di una clinica medica privata.

Keyword: Ingegneria del Software, Clinica privata, Ingegneria dei requisiti, Analisi, Specifica, Diagrammi, Casi d'uso, Matrice di mapping, Database, Progettazione concettuale, Entità, Relationship, Progettazione logica, Identificatore, Modello relazionale, Tabella, MVCS, UML, Classi, Attività, Python, MySQL, Manuale utente, SWOT Analysis

Indice

Elenco delle figure

Elenco delle tabelle

Elenco dei listati

Introduzione

Nel vasto panorama dell'ingegneria del software, uno dei task più stimolanti e complessi è la progettazione e lo sviluppo di software su misura, per soddisfare le esigenze specifiche di un'organizzazione. In questo contesto ci troviamo di fronte a una sfida insolita: creare un software dedicato alla gestione di una clinica privata immaginaria di nome "Casa Alfredo". Questo progetto non è semplicemente un esercizio accademico, ma un'opportunità per scoprire come le tecnologie informatiche possano migliorare l'efficienza operativa e l'esperienza dei pazienti all'interno di una clinica medica privata.

Le cliniche private affrontano sfide complesse, tra cui la gestione dei pazienti e del personale medico, nonché la pianificazione delle prenotazioni e delle visite; abbiamo, pertanto, intrapreso un approfondito processo di sviluppo, seguendo le prassi dettate dall'ingegneria del software.

Un software dedicato alla gestione di una clinica privata offre una serie di vantaggi significativi; esso permette di automatizzare le attività amministrative, contribuendo, così, a semplificare e alleggerire il carico di lavoro del personale medico. Inoltre, il software consente una migliore pianificazione delle prenotazioni, ottimizzando gli orari dei medici in base alla domanda dei pazienti.

Ciascuna sezione della progettazione e dello sviluppo del software, trattata nei vari capitoli, sarà inizialmente introdotta da una base teorica per garantire la comprensione dei principi fondamentali. Successivamente, i concetti illustrati verranno messi in pratica per assicurare la realizzazione efficace del software.

La presente tesi si compone di sette capitoli strutturati come di seguito specificato:

- Nel Capitolo 1 si esploreranno le caratteristiche, i vantaggi e gli svantaggi delle cliniche private reali, analizzando i relativi sistemi.
- Nel Capitolo 2 si descriverà la fase di analisi e specifica dei requisiti, fondamentale per l'ingegneria del software, per poi focalizzare l'attenzione sulla descrizione di requisiti funzionali e non funzionali e sull'utilizzo dei diagrammi dei casi d'uso.
- Nel Capitolo 3 si affronterà la progettazione della base di dati, coprendo le fasi di progettazione concettuale e di progettazione logica.
- Nel Capitolo 4 si discuterà della progettazione delle applicazioni, utilizzando i diagrammi delle classi e delle attività.
- Nel Capitolo 5 si esaminerà l'implementazione del software per poi introdurre il manuale utente, progettato per orientare nell'uso dell'applicazione.

- Nel Capitolo 6 si presenterà la SWOT Analysis, uno strumento che utilizzeremo per analizzare il software prodotto.
- Nel Capitolo 7 verranno tratte le conclusioni in merito a quanto svolto e saranno esaminati alcuni possibili sviluppi futuri.

CAPITOLO 1

I sistemi di gestione di cliniche

Nel presente capitolo esploreremo le caratteristiche generali delle cliniche private, i vantaggi che offrono rispetto alle istituzioni pubbliche e il modo in cui utilizzano tecnologie ingegneristiche e informatiche per migliorare l'assistenza medica e l'esperienza del paziente. Nel corso del capitolo, esamineremo alcuni esempi di cliniche private, partendo dall'analisi dei siti web, al fine di comprendere meglio come queste strutture stiano impiegando le tecnologie per offrire servizi più efficienti e personalizzati ai pazienti.

1.1 Premessa

Le cliniche private sono strutture presso le quali è possibile ottenere una vasta gamma di servizi medici e assistenza sanitaria a pagamento; sono distinte dalle istituzioni pubbliche per il loro carattere privatistico e per la possibilità di offrire servizi dedicati, tempi di attesa ridotti e accesso a tecnologie mediche all'avanguardia.

Le cliniche private sono gestite da società o individui privati, il che significa che il finanziamento delle operazioni proviene principalmente dai pazienti stessi. In tal modo, le cliniche private possono godere di una maggiore agilità organizzativa e di una rapida adozione di nuove tecnologie o servizi, in contrasto con le istituzioni pubbliche, che spesso sono vincolate dai finanziamenti governativi.

Uno dei vantaggi delle cliniche private è la ricca offerta di attività ambulatoriali e specializzazioni mediche, unitamente alla possibilità, da parte dell'utenza, di prenotare visite e ricevere diagnosi in tempi più rapidi rispetto al sistema sanitario pubblico. Tali cliniche operano in vari settori medici, come la cardiologia, la pneumologia, la gastroenterologia, l'endocrinologia e l'oculistica, offrendo servizi adatti sia a individui che ad aziende e consentendo di ottenere assistenza medica personalizzata, rispondente ai propri bisogni specifici.

Le cliniche private, come tutte le strutture sanitarie, si avvalgono di diverse tecnologie ingegneristiche e informatiche per fornire servizi di alta qualità, gestire i dati dei pazienti e migliorare l'efficienza delle operazioni. Di seguito, esploreremo alcuni aspetti rilevanti dell'utilizzo di tali tecnologie.

- *Sistemi di gestione dei dati*. Le cliniche private utilizzano software di gestione dei dati per registrare e mantenere tutte le informazioni relative ai pazienti e al personale medico, inclusi dati anagrafici, prenotazioni, storico delle visite, referti di laboratorio. Questi sistemi consentono un'organizzazione più efficiente dei dati e riducono il rischio di errori o duplicazioni nelle informazioni dei pazienti.

- *Dispositivi medici avanzati*. Le cliniche private spesso investono in dispositivi medici all'avanguardia, che possono includere macchinari per diagnosi avanzata, strumenti chirurgici specializzati e attrezzature per la riabilitazione. L'ingegneria biomedica svolge un ruolo fondamentale nella scelta, installazione e manutenzione di tali dispositivi per garantire il loro corretto funzionamento e la sicurezza dei pazienti.
- *Applicazioni mobile*. Molte cliniche private stanno abbracciando le tecnologie mobili, per consentire la riduzione dei tempi di consultazione dei dati clinici e per avere una visione completa dell'anamnesi del paziente in un'unica piattaforma. Le applicazioni mobile consentono ai pazienti di prenotare visite, ricevere notifiche, accedere alla propria cartella clinica e, talvolta, monitorare i parametri vitali, condividendoli direttamente con il personale medico, facilitando così il controllo delle condizioni di salute.
- *Sicurezza informatica e privacy dei dati*. La protezione dei dati dei pazienti è una priorità assoluta per le cliniche private, che devono adottare misure di sicurezza informatica per proteggere i dati sensibili dei pazienti da accessi non autorizzati o da perdite. Ciò include l'implementazione di protocolli di sicurezza, la crittografia dei dati e la formazione del personale riguardo alle pratiche di protezione dei dati.

In sintesi, l'ingegneria e l'informatica rivestono un ruolo fondamentale nella gestione delle cliniche private, contribuendo a migliorarne l'efficienza operativa, la qualità delle cure e l'esperienza complessiva dei pazienti.

Nel corso di questo capitolo esamineremo i sistemi informativi di cinque cliniche private, ovvero:

- Quisisana;
- San Francesco;
- Sanatrix;
- Santa Rita;
- Villa Margherita.

1.2 Analisi del sistema relativo alla clinica Quisisana

La casa di cura Quisisana, situata nel cuore di Roma, presenta un sistema con caratteristiche focalizzate sulla fornitura di servizi medici e diagnostici avanzati. Il sistema si concentra principalmente sull'esposizione delle caratteristiche generali della clinica ai potenziali pazienti, senza una rilevante interazione con gli stessi. Vengono illustrate accuratamente le diverse aree mediche e le attività ambulatoriali presenti all'interno della casa di cura, vengono fornite informazioni su personale medico con relative specializzazioni e vengono indicati i numeri di telefono dedicati ai vari servizi medici, per poter prenotare visite o ricoveri. Inoltre, il sistema mette a disposizione un servizio di hôtellerie-accoglienza per medici e pazienti.

Una peculiarità della clinica Quisisana è il servizio di diagnostica avanzato, supportato da tecniche di imaging di ultima generazione, che consente al personale medico di ottenere una diagnosi precisa ed efficace, garantendo così un trattamento ottimale ai pazienti.

Tuttavia, la limitata interazione con gli utenti potrebbe rappresentare un punto di debolezza del sistema. Da una parte questa scelta potrebbe essere intenzionale, per garantire efficienza e precisione nella comunicazione. D'altro canto, potrebbe essere utile valutare l'introduzione di nuove funzionalità interattive nel sistema, come un'applicazione mobile o l'introduzione di un portale destinato ai pazienti, per consentire a questi ultimi di prenotare

servizi medici in modo più diretto e di accedere a informazioni dettagliate riguardanti le diagnosi e le terapie seguite.

1.3 Analisi del sistema relativo alla clinica San Francesco

La clinica privata San Francesco, situata a Verona, presenta un sistema caratterizzato da un'ampia esposizione di informazioni riguardanti le attività ambulatoriali e il personale medico a disposizione dei pazienti.

Un aspetto significativo che contraddistingue questo sistema è l'implementazione di portali dedicati agli utenti, attraverso i quali essi possono effettuare prenotazioni e visualizzare i referti relativi alle visite eseguite all'interno della clinica.

Per accedere al portale per le prenotazioni, gli utenti devono registrarsi utilizzando i propri dati anagrafici. Una volta completata la registrazione, l'accesso può essere effettuato utilizzando un cellulare, o un indirizzo di posta elettronica, e una password. Gli utenti registrati possono così prenotare una visita, con la possibilità di scegliere l'ambulatorio, l'attività ambulatoriale e il personale medico desiderati, oltre a indicare una data approssimativa. Il sistema propone finestre settimanali con diverse opzioni di giorni e orari disponibili per le visite, fornendo anche il costo dell'attività ambulatoriale scelta. Il portale offre, inoltre, la possibilità di visualizzare le proprie prenotazioni e, se necessario, di disdire una visita pianificata ma non ancora effettuata. Questa funzionalità consente agli utenti di gestire in modo autonomo le prenotazioni effettuate, il tutto a portata di pochi click.

Al termine di una visita, ai pazienti verranno forniti uno username e una password per accedere a un altro portale specifico, tramite il quale poter visualizzare i referti medici.

La creazione di portali dedicati agli utenti consente loro di accedere in modo comodo ai referti medici, offrendo maggiore trasparenza e facilità nell'ottenere informazioni relative alle visite effettuate. Inoltre, permette di prenotare visite in modo rapido e intuitivo.

1.4 Analisi del sistema relativo alla clinica Sanatrix

Il sistema adottato dalla clinica privata Sanatrix, situata a Roma, presenta diverse caratteristiche simili a quelle precedentemente analizzate, come la descrizione di informazioni di carattere generale, quali gli ambulatori disponibili, le attività ambulatoriali offerte e il personale medico presente nella struttura. Come il precedente, il sistema include portali dedicati agli utenti, consentendo loro di creare e gestire le prenotazioni o anche di visualizzare i referti medici.

Una caratteristica che permette a questo sistema di distinguerlo dagli altri è la presenza di un'applicazione mobile, denominata "My Sanatrix", appositamente creata per la clinica. Attraverso di essa, il paziente può facilmente effettuare prenotazioni, gestirle e, se necessario, disdirle. Inoltre, l'applicazione consente al paziente di restare in contatto diretto con il personale medico, fornendo un canale di comunicazione per eventuali domande, dubbi o necessità di assistenza. Infine, l'app permette al paziente di accedere alla propria cartella clinica, consentendo una facile consultazione delle informazioni relative alla propria salute e ai trattamenti seguiti.

L'implementazione di un'applicazione mobile evidenzia l'attenzione della clinica Sanatrix verso l'adozione di tecnologie innovative per migliorare l'esperienza complessiva degli utenti.

1.5 Analisi del sistema relativo alla clinica Santa Rita

La clinica privata Santa Rita, situata a Vercelli, fa parte di un gruppo di cliniche che comprende sedi nelle maggiori città piemontesi, come Alessandria, Torino e Biella, oltre ad altre località, sia italiane che estere.

Il sistema implementato da Santa Rita è simile a quelli precedentemente analizzati, presentando un unico portale che consente agli utenti di gestire le prenotazioni e di accedere ai referti medici. Il portale unificato offre agli utenti un'esperienza uniforme e semplificata per la gestione delle cure mediche, consentendo loro di visualizzare o scaricare i referti.

Un segno distintivo di questo sistema è la varietà di opzioni disponibili per effettuare le prenotazioni. Oltre al portale online, i pazienti possono prenotare contattando un recapito telefonico o inviando un messaggio di testo a un numero dedicato.

La clinica Santa Rita ritiene importante comunicare il proprio impegno per la protezione dei dati personali dei pazienti, in conformità con il regolamento europeo 2016/679 sulla privacy. Questo atteggiamento proattivo verso la sicurezza dei dati è fondamentale per garantire la privacy e la riservatezza delle informazioni dei pazienti, creando un ambiente di fiducia e tranquillità per gli utenti.

1.6 Analisi del sistema relativo alla clinica Villa Margherita

La clinica privata Villa Margherita, situata a Roma e operante dal 1948, si distingue per una serie di caratteristiche rilevanti. La clinica offre servizi H24 nelle principali aree mediche, tra cui maternità, neurochirurgia e oculistica, garantendo una copertura continua per i pazienti che necessitano di cure in tali campi. Il sistema riesce a esporre accuratamente gli ambulatori e le specializzazioni di cui la clinica si compone, fornendo una dettagliata descrizione del personale medico coinvolto.

Analogamente ad altri sistemi precedentemente analizzati, la clinica Villa Margherita offre un unico portale dedicato agli utenti, che permette loro di gestire le prenotazioni e di accedere ai referti medici in modo integrato e pratico.

Un ulteriore punto di forza del sistema è rappresentato dalla presenza di un'applicazione mobile disponibile per dispositivi Android e iOS, che agevola il paziente nella visualizzazione dei propri dati personali, nella gestione delle prenotazioni e nell'accesso alla propria cartella clinica.

Tuttavia, è necessario evidenziare un'unica criticità riguardante la navigabilità del sito web della clinica. Alcuni link risultano non funzionanti o hanno un comportamento inatteso, mentre sezioni importanti, come quelle dedicate alla prenotazione delle visite o alla visualizzazione dei referti, compaiono a schermo solo per pochi istanti, rendendo difficoltosa l'interazione dell'utente con tali elementi. Sarebbe opportuno risolvere le criticità descritte per garantire una maggiore accessibilità da parte degli utenti.

CAPITOLO 2

Specifica e analisi dei requisiti

In questo capitolo esploreremo la fondamentale fase di specifica e analisi dei requisiti nello sviluppo di progetti software, focalizzando l'attenzione sulla descrizione di requisiti funzionali e non funzionali e sull'utilizzo dei diagrammi dei casi d'uso. I concetti introdotti verranno, poi, utilizzati per sviluppare il software per la clinica privata "Casa Alfredo". Infine, verrà introdotta la matrice di mapping, che collega i casi d'uso ai requisiti funzionali, agevolando la valutazione del soddisfacimento di questi ultimi.

2.1 Raccolta dei requisiti

La raccolta dei requisiti rappresenta una fase basilare nello sviluppo di un progetto software. Essa consiste nel raccogliere e documentare richieste e aspettative che il software deve soddisfare. Nel corso di questa fase, gli ingegneri del software, i progettisti e gli sviluppatori acquisiscono informazioni dagli stakeholder del sistema attraverso varie tecniche, tra cui:

- *Interviste*: incontri sia formali che informali, finalizzati alla scoperta di informazioni, comunicando direttamente con gli stakeholder, gli utenti finali e gli esperti del dominio. Le interviste sono spesso condotte in modo individuale, o possono coinvolgere più intervistatori e/o intervistati.
- *Focus group*: riunioni di stakeholder esperti del settore, per conoscere le relative aspettative e le opinioni sul prodotto da sviluppare.
- *Brainstorming*: tecnica utilizzata per generare e raccogliere diverse idee relative ai requisiti del progetto e del prodotto.
- *Questionari e sondaggi*: serie di domande scritte, studiate per raccogliere rapidamente informazioni da un ampio numero di partecipanti. Queste tecniche sono molto efficaci quando sono necessari tempi di risposta rapidi nella raccolta delle caratteristiche del sistema.

Una corretta raccolta dei requisiti rappresenta la base per la fase di progettazione del software, in quanto fornisce le informazioni necessarie per definire l'architettura del sistema e guidare il processo di sviluppo.

La raccolta dei requisiti riguardante la progettazione e lo sviluppo del software per la clinica privata "Casa Alfredo" è stata eseguita intervistando uno stakeholder della clinica. Di seguito, è riportata la descrizione in linguaggio naturale dell'intervista.

2.1.1 Descrizione in linguaggio naturale

Il software da sviluppare consiste in un sistema informativo per la gestione della clinica privata "Casa Alfredo".

In particolare, l'applicativo deve poter essere utilizzato dai pazienti della clinica per prenotare visite presso uno specifico ambulatorio, per accedere alle prenotazioni già effettuate, per visualizzare la propria cartella clinica e per gestire i propri dati.

Inoltre, l'applicazione deve poter essere utilizzata anche dal personale medico per accedere ai propri dati, per visualizzare il calendario delle prenotazioni, per accedere alla cartella clinica del paziente e per aggiornarla dopo ogni visita. Nello specifico, le prenotazioni possono essere effettuate solo ed esclusivamente da pazienti registrati.

Ogni prenotazione deve comprendere un codice identificativo, la data e ora della stessa, la data e l'ora effettiva della visita presso l'ambulatorio di interesse, il nome dell'ambulatorio, l'attività ambulatoriale, l'operatore assegnato, ed, infine il costo della visita. Inoltre, ciascuna prenotazione può essere disdetta fino a 24 ore prima della data effettiva dell'appuntamento in sede. In caso di prenotazione disdetta, eventuali pazienti possono prenotare la stessa visita ambulatoriale nella fascia oraria resa così disponibile. In aggiunta a quanto appena espresso, l'applicazione permetterà di visualizzare solo ed esclusivamente i giorni e gli orari in cui è possibile prenotare una visita e, di conseguenza, negherà l'accesso ai giorni e agli orari già prenotati. Infine, il limite massimo temporale di prenotazione è di un anno a partire dalla data e dall'ora in cui si accede al servizio di prenotazione.

La clinica privata "Casa Alfredo" gestisce sei ambulatori, rispettivamente noti come:

- Medicina generale cardiologica;
- Endocrinologia;
- Urologia/Andrologia;
- Oculistica;
- Ortopedia;
- Dermatologia.

In particolare, ciascuno degli ambulatori dispone di specifici orari di apertura e di chiusura, come di seguito riportato:

- **Medicina generale cardiologica**:
  - lunedì, mercoledì, venerdì:
    - 07:00 - 13:00;
    - 15:00 - 19:00.
- **Endocrinologia**:
  - martedì, giovedì:
    - 07:00 - 12:00;
    - 15:00 - 17:00.
- **Urologia/Andrologia**:
  - mercoledì, giovedì, venerdì:
    - 11:00 - 13:00;

        - 15:00 - 18:00.

- **Oculistica**:
    - lunedì, mercoledì, sabato:
        - 07:00 - 13:00;
        - 15:00 - 17:00.

- **Ortopedia**:
    - martedì, venerdì:
        - 09:00 - 14:00;
        - 16:00 - 19:00.

- **Dermatologia**:
    - martedì:
        - 07:00 - 13:00;
        - 15:00 - 19:00.

Inoltre, il sistema gestisce diversamente i dati relativi alla tipologia di utente. La registrazione dei pazienti richiede i seguenti dati:

- nome;
- cognome;
- data di nascita;
- età;
- sesso;
- città di nascita;
- provincia di nascita;
- codice fiscale;
- e-mail e password, con le quali il paziente può accedere al sistema.

Il personale medico della clinica è previamente registrato nel sistema; per ciascun componente vengono registrati i seguenti dati:

- nome;
- cognome;
- data di nascita;
- età;
- sesso;
- città di nascita;
- provincia di nascita;

- e-mail;
- codice fiscale;
- ambulatorio;
- codice identificativo e password, con i quali l'operatore può accedere al sistema.

Le attività ambulatoriali prevedono le seguenti tariffe e durate:

- **Medicina generale cardiologica**:
  - Specializzazione cardiovascolare:
    - Ecocardiografia - €110 - 30 minuti;
    - ECG basale o da sforzo - €60 - 15 minuti;
    - Holter cardiaco - €62 - 15 minuti;
    - Holter pressorio - €50 - 15 minuti;
    - Visita cardiologica - €125 - 30 minuti;
    - Angiologia - €50 - 20 minuti.
  - Specializzazione pneumologica:
    - Visita pneumologica - €50 - 20 minuti.
  - Specializzazione gastroenterologica:
    - Visita gastroenterologica - €150 - 20 minuti;
    - Gastroscopia - €160 - 15 minuti;
    - Colonscopia - €210 - 30 minuti.
  - Specializzazione neurologica:
    - Visita neurologica - €100 - 30 minuti.
- **Endocrinologia**:
  - Visita endocrinologica - €130 - 40 minuti;
  - Visita specialistica - €50 - 30 minuti.
- **Urologia/Andrologia**:
  - Visita urologica - €100 - 20 minuti;
  - Visita andrologica - €120 - 20 minuti;
  - Cistoscopia - €300 - 10 minuti;
  - Ecografia renale - €40 - 20 minuti;
  - Ecografia vescicale - €40 - 20 minuti;
  - Ecografia prostatica sovrapubica - €40 - 20 minuti.
- **Oculistica**:
  - Visita oculistica - €100 - 30 minuti;
  - Visita ortottica - €70 - 30 minuti;
  - Studio del campo visivo - €150 - 30 minuti;
  - Perimetria computerizzata - €130 - 20 minuti;
  - Pachimetria corneale - €100 - 15 minuti;

- Tomografia corneale - €100 - 15 minuti.

- **Ortopedia**:
  - Visita ortopedica - €120 - 20 minuti;
  - Iniezioni intra - articolari - €60 - 10 minuti;
  - Trattamento riabilitativo ortopedico - €45 - 1 ora.

- **Dermatologia**:
  - Visita dermatologica - €80 - 30 minuti;
  - Biopsie cutanee - €150 - 30 minuti;
  - Mappatura nevi - €120 - 30 minuti;
  - Crioterapia - €130 - 10 minuti;
  - Dermatoscopia - €80 - 40 minuti.

In sintesi, ciascuna visita prevede un ID, il nome dell'ambulatorio di riferimento, il costo della visita e la durata della visita espressa in minuti.

2.1.2 Glossario dei termini

Nella Tabella 2.1 è riportato il glossario dei termini; esso memorizza tutte le parole contenute nella descrizione in linguaggio naturale, che possono creare ambiguità con la relativa spiegazione. È importante notare la distinzione tra "Business" e "Tecnico" sotto la voce "Tipo":

- *Business*. I termini contrassegnati come "Business" sono legati agli aspetti funzionali, amministrativi e gestionali della clinica.
- *Tecnico*. I termini contrassegnati come "Tecnico" si riferiscono agli aspetti tecnologici e informatici relativi all'applicativo da sviluppare.

Termine	Descrizione	Tipo	Sinonimi
Clinica	Struttura sanitaria che si concentra principalmente sulla cura dei pazienti ambulatoriali	Business	Sede
Ambulatorio	Locale attrezzato per visite mediche e cure specialistiche che non richiedono una degenza ospedaliera	Business	Nessuno
Prenotazione	Atto mediante il quale viene fissato, in una data stabilita, un appuntamento con un medico	Business/Tecnico	Nessuno
Paziente	Chi è affidato alle cure di un medico	Business/Tecnico	Utente
Personale medico	Chi opera all'interno della clinica	Business/Tecnico	Operatore medico, utente
Cartella clinica	Documento che raccoglie le informazioni di tipo medico ed infermieristico riguardanti il paziente	Business/Tecnico	Nessuno

Visita ambulatoriale	Analisi relativa alla salute del paziente condotta dal medico di uno specifico ambulatorio	Business	Attività ambulatoriale

Tabella 2.1: Glossario dei termini

2.2 Analisi dei requisiti

L'analisi dei requisiti è la fase che segue la raccolta dei requisiti, volta a identificare, comprendere e documentare in modo accurato i requisiti che il software da sviluppare deve soddisfare. I requisiti si distinguono in funzionali e non funzionali.

- *Requisiti funzionali*: descrivono le specifiche azioni che il sistema deve essere in grado di eseguire, delineando le funzionalità, i comportamenti e le interazioni con gli utenti. I requisiti funzionali sono le caratteristiche che influenzano direttamente l'esperienza dell'utente e il comportamento del sistema.
- *Requisiti non funzionali*: si concentrano su qualità, prestazioni e restrizioni che il sistema deve rispettare, ma che non sono direttamente legate alle funzionalità specifiche. Questa tipologia di requisiti riguarda aspetti come la sicurezza, la scalabilità, la velocità, l'affidabilità, l'usabilità e altri attributi che influenzano la qualità globale del sistema.

Verranno ora presentati e dettagliati i requisiti funzionali e non funzionali identificati mediante l'analisi della descrizione in linguaggio naturale, precedentemente condotta.

2.2.1 Descrizione requisiti funzionali

I requisiti funzionali possono essere organizzati in macroaree; queste sono:

- utente;
- paziente;
- pesonale medico.

Questa suddivisione è rappresentata graficamente in Figura 2.1.

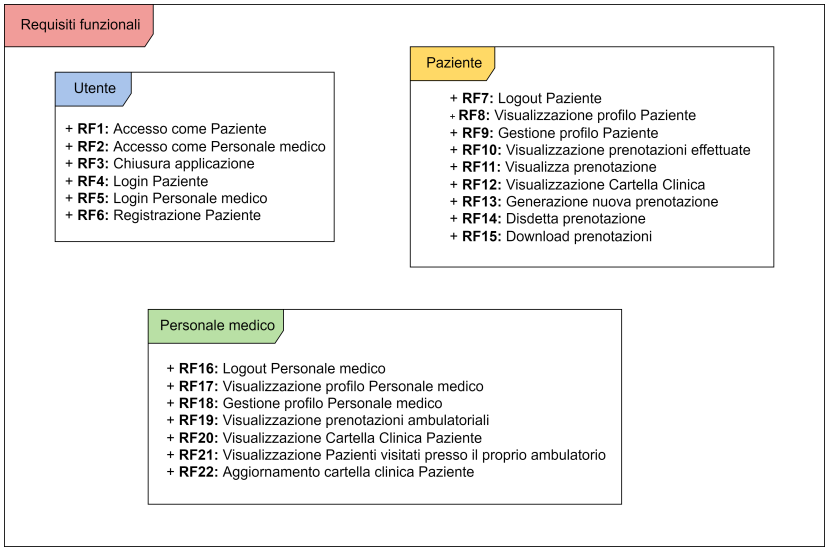

Figura 2.1: Requisiti funzionali raggruppati in macroaree

Descriviamo, nella Tabella 2.2, ogni singolo requisito funzionale.

Requisito	Descrizione
RF1: Accesso come paziente	Il sistema, al primo avvio, dovrà consentire all'utente di accedere come paziente
RF2: Accesso come personale medico	Il sistema, al primo avvio, dovrà consentire all'utente di accedere come personale medico
RF3: Chiusura applicazione	Il sistema dovrà consentire all'utente di chiudere l'applicazione
RF4: Login paziente	Il sistema dovrà consentire l'autenticazione del paziente fornendo e-mail e password
RF55: Login personale medico	Il sistema dovrà consentire l'autenticazione del personale medico fornendo codice identificativo e password
RF6: Registrazione paziente	Il sistema dovrà consentire la registrazione di un nuovo paziente qualora egli non fosse già registrato
RF7: Logout paziente	Il sistema dovrà consentire al paziente, una volta effettuato l'accesso, di eseguire un logout
RF8: Visualizzazione profilo paziente	Il sistema dovrà consentire al paziente di visualizzare le proprie informazioni personali
RF9: Gestione profilo paziente	Il sistema dovrà consentire al paziente di modificare le proprie informazioni personali
RF10: Visualizzazione prenotazioni effettuate	Il sistema dovrà consentire al paziente di visualizzare le prenotazioni da lui effettuate
RF11: Visualizzazione prenotazione	Il sistema dovrà consentire al paziente di visualizzare la prenotazione scelta dalla lista delle prenotazioni effettuate
RF12: Visualizzazione cartella clinica	Il sistema dovrà consentire al paziente di visualizzare la propria cartella clinica
RF13: Generazione nuova prenotazione	Il sistema dovrà consentire al paziente di generare una nuova prenotazione, scegliendo ambulatorio, attività ambulatoriale, data e orario
RF14: Disdetta prenotazione	Il sistema dovrà consentire al paziente la possibilità di annullare una prenotazione effettuata entro le 24 ore precedenti all'appuntamento fissato
RF15: Download prenotazioni	Il sistema dovrà consentire al paziente di scaricare le proprie prenotazioni in formato PDF
RF16: Logout personale medico	Il sistema dovrà consentire al personale medico, una volta effettuato l'accesso, di eseguire un logout
RF17: Visualizzazione profilo personale medico	Il sistema dovrà consentire al personale medico di visualizzare le proprie informazioni personali
RF18: Gestione profilo personale medico	Il sistema dovrà consentire al personale medico di modificare le proprie informazioni personali
RF19: Visualizzazione prenotazioni ambulatoriali	Il sistema dovrà consentire al personale medico di visualizzare le prenotazioni effettuate dai pazienti presso il proprio ambulatorio
RF20: Visualizzazione cartella clinica paziente	Il sistema dovrà consentire al Personale medico di visualizzare la cartella clinica dei pazienti attraverso le prenotazioni effettuate presso il proprio ambulatorio

RF21: Visualizzazione pazienti visitati presso il proprio ambulatorio	Il sistema dovrà consentire al personale medico di visualizzare la lista dei pazienti visitati presso il proprio ambulatorio
RF22: Aggiornamento cartella clinica paziente	Il sistema dovrà consentire al personale medico di aggiornare la cartella clinica dei pazienti visitati presso il proprio ambulatorio

Tabella 2.2: Descrizione dei requisiti funzionali

2.2.2 Descrizione requisiti non funzionali

Nella Tabella 2.3 vengono descritti i requisiti non funzionali.

Requisito	Descrizione
RNF1: Implementazione	Il sistema dovrà essere realizzato in tecnologia Python 3
RNF2: Modularità	Il sistema dovrà essere composto da moduli (classi)
RNF3: Documentazione	Ogni modulo del sistema dovrà essere documentato
RNF4: Portabilità	Il sistema potrà essere eseguito su sistemi operativi diversi
RNF5: Interfaccia grafica	Il sistema dovrà essere dotato di interfaccia grafica
RNF6: User-friendly	L'interfaccia grafica del sistema dovrà essere facilmente utilizzabile da ogni utente
RNF7: ID	Il sistema non dovrà consentire l'impiego di ID già in uso
RNF8: Limite prenotazione	Il sistema non dovrà consentire la prenotazione di un'attività ambulatoriale oltre un anno
RNF9: Limite disdetta	Il sistema non dovrà consentire l'annullamento di una prenotazione nelle 24 ore precedenti all'appuntamento fissato
RNF10: Comprensibilità	Il sistema dovrà essere comprensibile per il tipo di utenti per i quali è progettato
RNF11: Usabilità	Il sistema dovrà essere usabile per il tipo di utenti per i quali è progettato

Tabella 2.3: Descrizione dei requisiti non funzionali

2.3 Documentazione dei requisiti

In questa sezione saranno delineati in modo dettagliato i requisiti funzionali attraverso i diagrammi dei casi d'uso.

Un diagramma dei casi d'uso è un tipo di diagramma UML che descrive le interazioni tra un attore e un sistema. Gli attori sono le persone o le cose che interagiscono con il sistema. I casi d'uso sono le attività che l'attore può svolgere con il sistema.

I diagrammi dei casi d'uso, data la loro semplice rappresentazione, possono essere utilizzati per comunicare i requisiti del sistema agli sviluppatori, ai tester e agli utenti finali.

L'impiego dei diagrammi dei casi d'uso porta a diversi vantaggi:

- *Chiarezza nei requisiti*. I diagrammi dei casi d'uso aiutano a garantire che i requisiti del sistema siano compresi da tutti i membri del team di sviluppo, evitando equivoci e ambiguità. Ciò può portare a un sistema più conforme ai requisiti e ad un minor numero di problemi dopo il rilascio.

- *Individuazione e risoluzione di problemi*. I diagrammi dei casi d'uso sono strumenti efficaci per individuare e risolvere criticità del sistema, contribuendo, così, alla prevenzione di problemi e all'abbattimento dei costi di realizzazione e manutenzione.
- *Comunicazione migliorata*. I diagrammi dei casi d'uso facilitano una comunicazione più efficace tra il team di sviluppo e gli utenti finali. Ciò può contribuire ad assicurare il soddisfacimento delle necessità degli utenti e la progettazione di un sistema user-friendly.
- *Riduzione dei costi di sviluppo*. I diagrammi dei casi d'uso contribuiscono a contenere i costi di sviluppo attraverso una maggiore comprensione delle esigenze e dei requisiti, evitando modifiche onerose in fasi avanzate del progetto.

Incorporando i diagrammi dei casi d'uso nella fase di analisi dei requisiti, si ottimizza la pianificazione, la progettazione e l'implementazione del sistema, generando risultati più soddisfacenti in tutte le fasi coinvolte.

2.3.1 Diagramma dei casi d'uso: Attori

Gli attori che interagiscono direttamente con il sistema sono gli utenti generici, i pazienti ed il personale medico, che verranno rappresentati come in Figura 2.2.

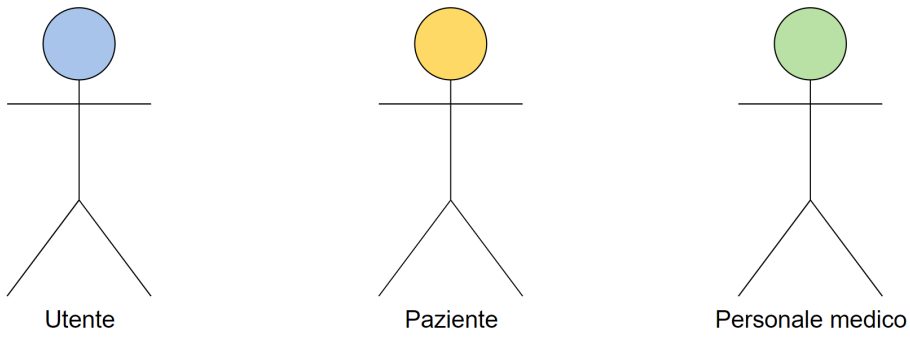

Figura 2.2: Attori del sistema

2.3.2 Diagramma dei casi d'uso: Utente

In Figura 2.3 viene illustrato il diagramma dei casi d'uso dell'utente generico, che definisce l'accesso, la registrazione e la gestione delle sessioni per i pazienti e per il personale medico. L'utente può accedere come paziente selezionando la tipologia di utente e avviando la procedura di login mediante l'inserimento delle credenziali. Se già registrato, il paziente accede al sistema; in caso contrario può registrarsi, inserendo le informazioni richieste. Analogamente, il personale medico può accedere attraverso una procedura di login dedicata. In entrambi i casi, il sistema verifica le credenziali e consente l'accesso, o lo nega in caso di errore. Il ritorno alla home page dell'utente è disponibile sia nella schermata di login del paziente che in quella del personale medico. Infine, l'utente può uscire dall'applicazione attivando la procedura apposita.

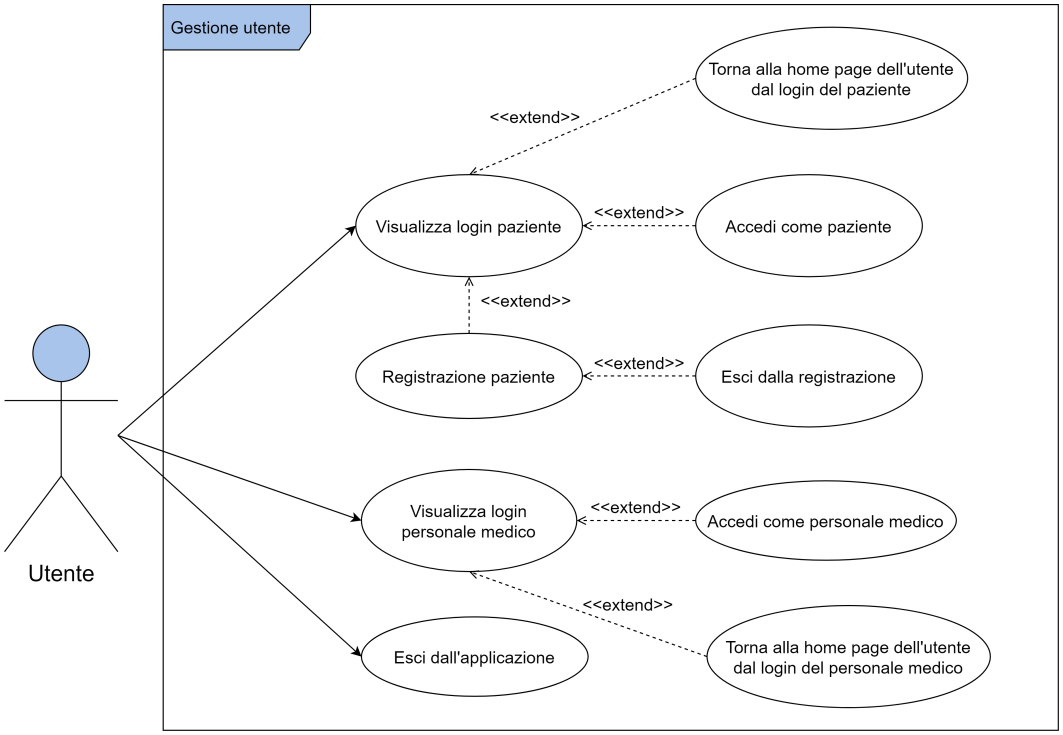

Figura 2.3: Gestione dell'utente da parte del sistema

2.3.3 Diagramma dei casi d'uso: Paziente

I diagrammi dei casi d'uso dei pazienti possono essere raggruppati graficamente come in Figura 2.4.

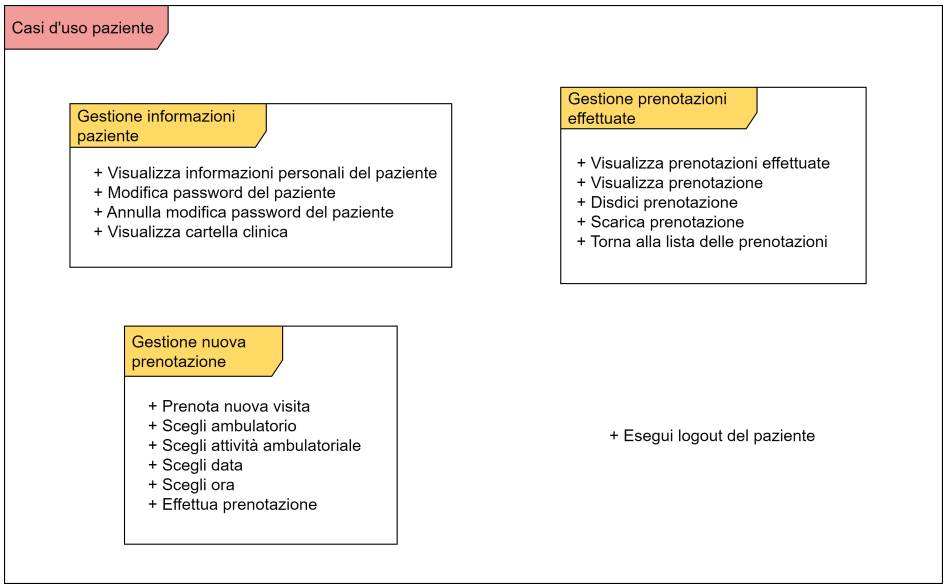

Figura 2.4: Casi d'uso del paziente

Gestione delle informazioni del paziente

In Figura 2.5 viene illustrato il diagramma dei casi d'uso di un paziente autenticato per la gestione delle proprie informazioni. Il paziente può visualizzare le informazioni personali

e da qui modificare la password, rispettando determinati requisiti. Il paziente, una volta avviata la procedura di modifica della password, può annullare tale operazione. Inoltre, il paziente può visualizzare la propria cartella clinica.

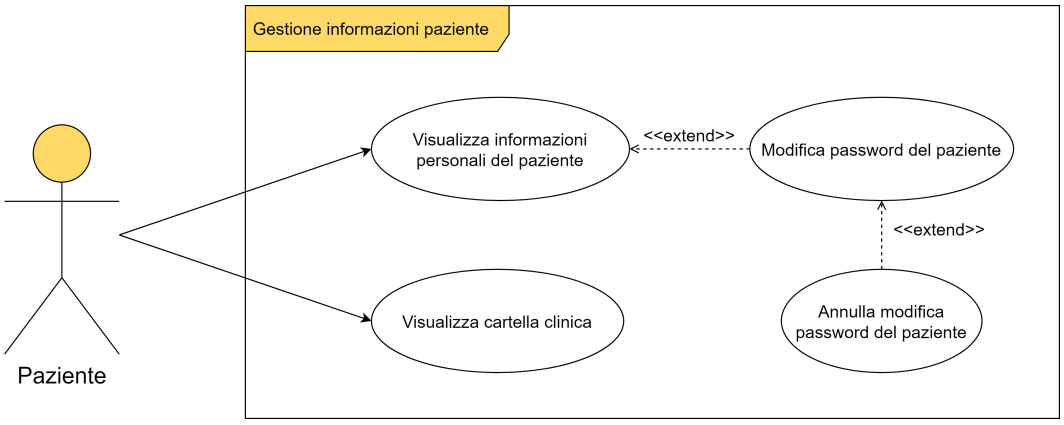

Figura 2.5: Gestione delle informazioni del paziente

Gestione delle prenotazioni effettuate

In Figura 2.6 viene illustrato il diagramma dei casi d'uso di un paziente autenticato per la gestione delle prenotazioni effettuate. Il paziente può visualizzare le prenotazioni effettuate e da qui visualizzare dettagli specifici delle stesse. Nella schermata relativa ai dettagli di una singola prenotazione, l'utente può disdire tale prenotazione, scaricare il pdf relativo e tornare alla lista delle prenotazioni effettuate.

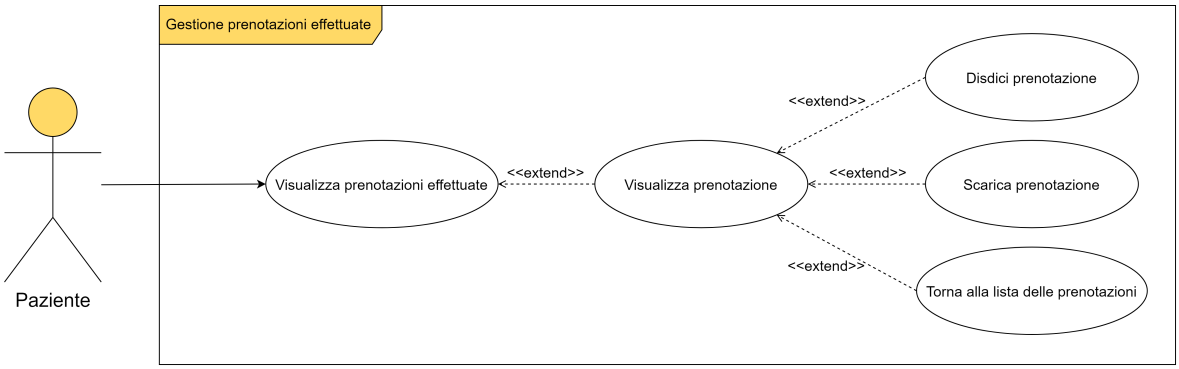

Figura 2.6: Gestione delle prenotazioni effettuate

Gestione di una nuova prenotazione

In Figura 2.7 viene illustrato il diagramma dei casi d'uso di un paziente autenticato per la gestione di una nuova prenotazione. Inizialmente, il paziente avvia la procedura di prenotazione, selezionando l'ambulatorio, l'attività ambulatoriale desiderata, la data e l'orario. Successivamente, il paziente conferma la registrazione della prenotazione. Se i dati inseriti sono corretti, il sistema memorizza la prenotazione e mostra un messaggio di conferma. Nel caso in cui si verifichino errori, il sistema notifica il paziente attraverso un messaggio e fornisce la possibilità di apportare correzioni alle informazioni fornite.

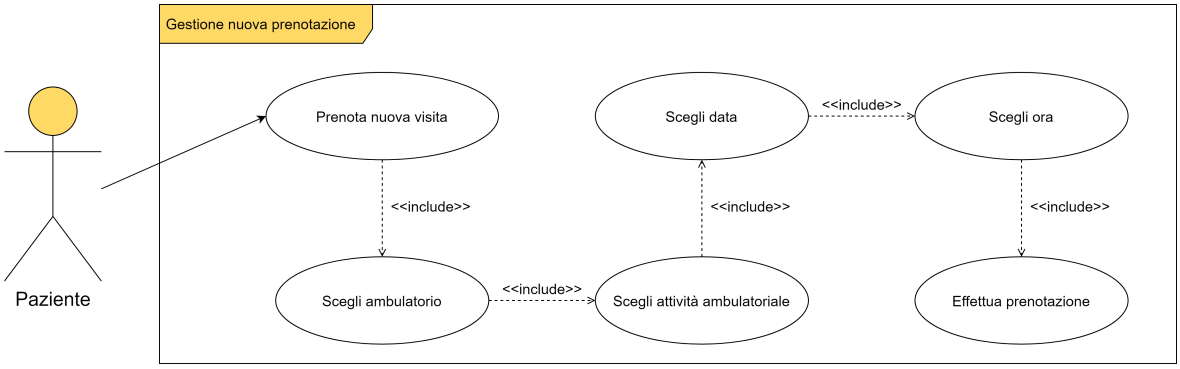

Figura 2.7: Gestione di una nuova prenotazione

Altro

In Figura 2.8 viene illustrato il diagramma dei casi d'uso di un paziente autenticato per gli scenari che non rientrano nelle macroaree appena descritte. Una volta autenticato, il paziente può eseguire il logout in qualunque punto dell'applicazione.

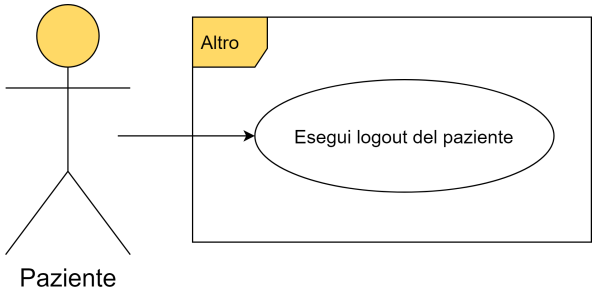

Figura 2.8: Altri casi d'uso del paziente

2.3.4 Diagramma dei casi d'uso: Personale medico

I diagrammi dei casi d'uso del personale medico possono essere raggruppati graficamente come in Figura 2.9.

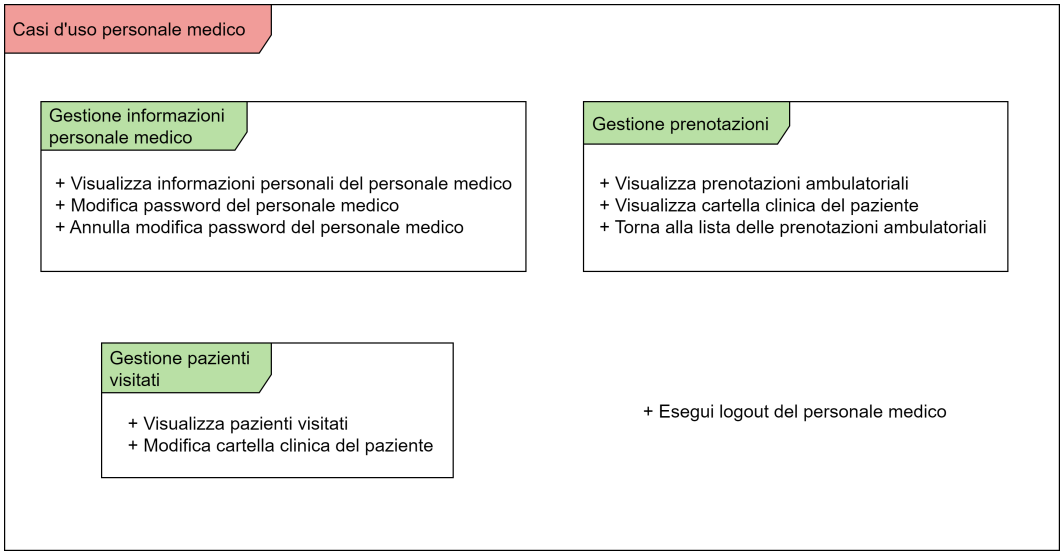

Figura 2.9: Casi d'uso del personale medico

Gestione delle informazioni del personale medico

In Figura 2.10 viene illustrato il diagramma dei casi d'uso del personale medico autenticato per la gestione delle proprie informazioni. Il personale medico può visualizzare le informazioni personali e da qui modificare la password, rispettando determinati requisiti. Il personale medico, una volta avviata la procedura di modifica della password, può annullare tale operazione.

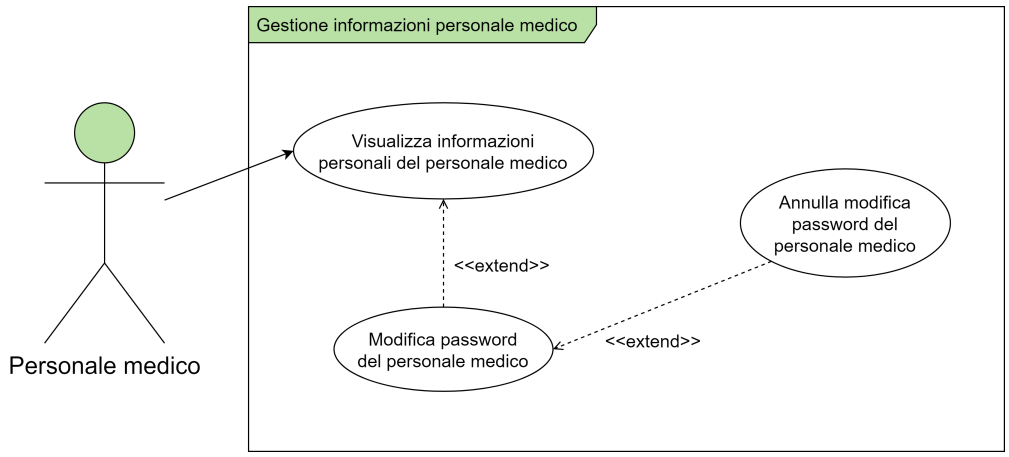

Figura 2.10: Gestione delle informazioni del personale medico

Gestione delle prenotazioni

In Figura 2.11 viene illustrato il diagramma dei casi d'uso del personale medico autenticato per la gestione delle prenotazioni. Il personale medico può consultare le prenotazioni effettuate relative al proprio ambulatorio e accedere alle cartelle cliniche dei singoli pazienti. Inoltre, nella schermata in cui si visualizza la cartella clinica di uno specifico paziente, è possibile tornare alla lista delle prenotazioni ambulatoriali.

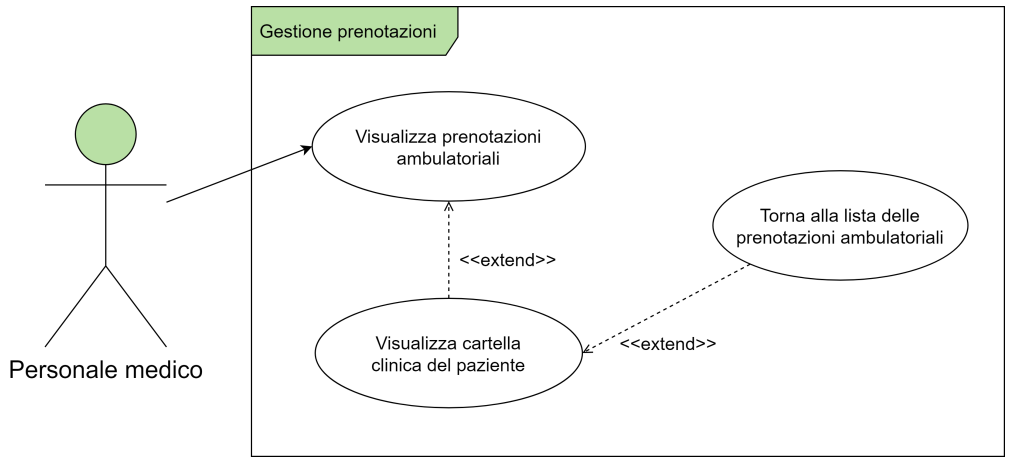

Figura 2.11: Gestione delle prenotazioni

Gestione dei pazienti visitati

In Figura 2.12 viene illustrato il diagramma dei casi d'uso del personale medico autenticato per la gestione dei pazienti visitati. Il personale medico ha la possibilità di accedere

alla lista delle visite effettuate, e successivamente, aggiornare le cartelle cliniche dei pazienti, includendo dettagli delle visite eseguite.

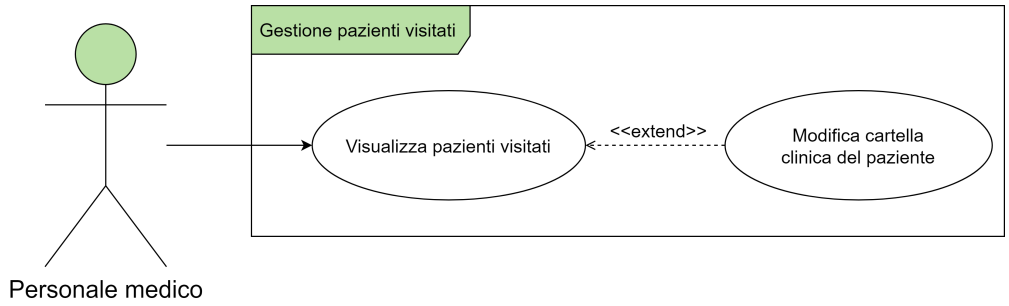

Figura 2.12: Gestione dei pazienti visitati

Altro

In Figura 2.13 viene illustrato il diagramma dei casi d'uso del personale medico autenticato per gli scenari che non rientrano nelle macroaree appena descritte. Una volta autenticato, il personale medico può eseguire il logout in qualunque punto dell'applicazione.

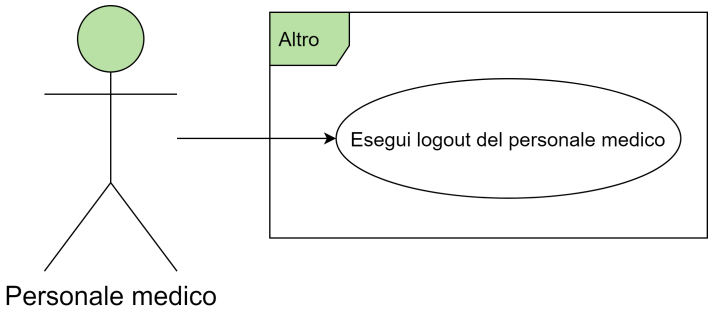

Figura 2.13: Altri casi d'uso del personale medico

2.4 Matrice di mapping

La matrice di mapping è uno strumento utilizzato per mettere in relazione elementi diversi del processo di sviluppo software. Essa, nel nostro caso, viene utilizzata per correlare i casi d'uso con i requisiti funzionali. In questo modo, il progettista riesce a visualizzare con maggiore facilità se i casi d'uso rappresentati sono sufficienti a garantire il soddisfacimento di tutti i requisiti funzionali descritti.

Nella Tabella 2.4 è rappresentata la matrice di mapping relativa all'utente generico.

	RF1	RF2	RF3	RF4	RF5	RF6
Visualizza login paziente	X					
Accedi come paziente				X		
Torna alla home page utente dal login del paziente	X					
Registrazione paziente						X
Esci dalla registrazione						X
Visualizza login personale medico		X				

Accedi come personale medico					X	
Torna alla home page utente dal login del personale medico		X				
Esci dall'applicazione			X			

Tabella 2.4: Matrice di mapping relativa all'utente

Nella Tabella 2.5 è rappresentata la matrice di mapping relativa al paziente.

	RF7	RF8	RF9	RF10	RF11	RF12	RF13	RF14	RF15
Visualizza informazioni personali del paziente		X							
Modifica password del paziente			X						
Annulla modifica password del paziente			X						
Visualizza cartella clinica						X			
Visualizza prenotazioni effettuate				X					
Visualizza prenotazione					X				
Disdici prenotazione								X	
Scarica prenotazione									X
Torna alla lista delle prenotazioni					X				
Prenota nuova visita							X		
Scegli ambulatorio							X		
Scegli attività ambulatoriale							X		
Scegli data							X		
Scegli ora							X		
Effettua prenotazione							X		
Esegui logout del paziente	X								

Tabella 2.5: Matrice di mapping relativa al paziente

Nella tabella 2.6 è rappresentata la matrice di mapping relativa al personale medico.

	RF16	RF17	RF18	RF19	RF20	RF21	RF22
Visualizza informazioni personali del personale medico		X					
Modifica password del personale medico			X				
Annulla modifica password del personale medico			X				
Visualizza prenotazioni ambulatoriali				X			
Visualizza cartella clinica del paziente					X		
Torna alla lista delle prenotazioni ambulatoriali					X		
Visualizza pazienti visitati						X	

Modifica cartella clinica del paziente							X
Esegui logout del personale medico	X						

Tabella 2.6: Matrice di mapping relativa al personale medico

CAPITOLO 3

Progettazione della base di dati

In questo capitolo verrà affrontata la progettazione della base di dati del software per la clinica privata "Casa Alfredo", in tutte le sue sfaccettature. Partiremo dalla progettazione concettuale, che rappresenta l'inizio del processo di progettazione, utilizzando il modello E-R per rappresentare astrattamente entità, attributi e relazioni chiave. Successivamente, procederemo con la ristrutturazione dello schema concettuale, applicando le regole proposte dalla progettazione logica, per ottenere una struttura più dettagliata e implementabile. Termineremo, poi, con la traduzione del modello concettuale nel modello logico, creando tabelle e definendo le relazioni tra di esse.

3.1 Introduzione alla progettazione della base di dati

La progettazione della base di dati costituisce il processo su cui poggia gran parte delle moderne applicazioni e dei sistemi informativi. L'obiettivo principale di questa fase è creare una struttura che faciliti l'archiviazione, l'organizzazione e il recupero dei dati in modo efficiente e coerente. Una progettazione ben eseguita non solo ottimizza le prestazioni, ma migliora anche la qualità e l'integrità dei dati nel tempo.

Di seguito sono elencati alcuni dei concetti fondamentali che una base di dati, prodotto finale della fase di progettazione, deve implementare.

- *Efficienza*. La progettazione deve essere efficiente in termini di spazio e di tempo. Ciò richiede che i dati siano memorizzati in modo da poter essere facilmente recuperati e aggiornati.
- *Coerenza*. La progettazione deve garantire che i dati siano coerenti. Ciò comporta che i dati siano accurati e aggiornati, prevenendo, così, errori e inconsistenze.
- *Affidabilità*. La progettazione deve garantire l'affidabilità, cioè la capacità del sistema di conservare intatto il contenuto del database (o almeno di permetterne la ricostruzione) in caso di malfunzionamenti hardware e software.
- *Sicurezza*. La progettazione deve garantire la sicurezza dei dati. Questo implica l'implementazione di meccanismi che proteggano le informazioni da accessi non autorizzati, garantendo la riservatezza dei dati.

Nel corso di questo capitolo, esploreremo le diverse fasi coinvolte nella progettazione della base di dati del software della clinica privata "Casa Alfredo".

3.2 Progettazione concettuale

Nell'ambito della progettazione concettuale, si delineano le strutture essenziali che contribuiranno alla realizzazione della base di dati finale. Nel corso del paragrafo descriveremo i passi fondamentali della progettazione concettuale, basandoci sull'analisi del sistema, effettuata nel precedente capitolo.

Inizialmente, procederemo con l'identificazione delle entità essenziali, che costituiscono le componenti principali dei dati da gestire. Successivamente, identificheremo gli attributi delle entità essenziali, passando, poi, all'identificazione delle relazioni fondamentali, in cui si stabilisce come le varie entità si legano tra loro.

Una volta definite le entità con i loro attributi e le relazioni che le collegano, si può procedere con la modellazione dello schema concettuale scheletro, fornendo una struttura di base da cui partire. Durante questo processo, possono emergere necessità di modifiche alla struttura delle entità, dovute al modo in cui esse interagiscono.

In seguito, procederemo a perfezionare ulteriormente il modello, attraverso la creazione dello schema concettuale completo, che rappresenterà il risultato finale della fase di progettazione concettuale.

Infine, considereremo i vincoli non esprimibili, ossia aspetti che influenzano il modo in cui i dati interagiscono, ma che non possono essere rappresentati esplicitamente nello schema concettuale.

La definizione dello schema concettuale sarà realizzata attraverso l'uso del modello Entità-Relazione (E-R). Quest'ultimo è un metodo grafico usato per descrivere la struttura dei dati e le relazioni tra di essi, in modo intuitivo e comprensibile sia agli sviluppatori che agli stakeholder.

Nello schema E-R, le entità sono rappresentate come rettangoli, a cui sono associati gli attributi dell'entità considerata. Le relazioni sono rappresentate da rombi che, tramite linee, collegano due o più entità.

Strategie utilizzate

Nella definizione dello schema concettuale viene adottata una strategia progettuale mista, fondata sui seguenti approcci:

- *Strategia Top-Down*: consiste nella produzione di uno schema concettuale mediante raffinamenti successivi, partendo dall'individuazione degli elementi essenziali e aggiungendo progressivamente dettagli. In sintesi, si procede dal livello generale a quello particolare.
- *Strategia Bottom-Up*: consiste nel descrivere, inizialmente, le singole componenti del sistema attraverso uno schema concettuale dettagliato. Successivamente, queste parti vengono integrate in maniera graduale per creare una prospettiva più ampia e complessa. In pratica, la progettazione procede dal livello specifico (bottom) al livello generale (up).

3.2.1 Identificazione delle entità essenziali

Mediante l'attività di analisi del sistema, è stato possibile individuare le seguenti entità:

- *Comune*. Un'istanza dell'entità "Comune" descrive il comune di nascita, che può essere associato ad uno o più utenti registrati nel sistema.

- *Ambulatorio*. Un'istanza dell'entità "Ambulatorio" descrive una specifica area della clinica.
- *Specializzazione*. Un'istanza dell'entità "Specializzazione" descrive uno specifico settore di un ambulatorio.
- *Attività ambulatoriale*. Un'istanza dell'entità "Attività ambulatoriale" descrive una specifica prestazione medica.
- *Paziente*. Un'istanza dell'entità "Paziente" descrive un singolo utente registrato nel sistema come paziente.
- *Personale medico*. Un'istanza dell'entità "Personale medico" descrive un singolo utente previamente registrato nel sistema come operatore medico.
- *Prenotazione*. Un'istanza dell'entità "Prenotazione" descrive una singola visita fissata da un paziente registrato nel sistema.

3.2.2 Identificazione degli attributi delle entità essenziali

In questa fase, si considerano le singole entità precedentemente elencate e le si arricchiscono di attributi, utilizzando, quindi, la strategia Top-Down.

Entità Comune

In Figura 3.1 è illustrata graficamente l'entità "Comune", che possiede i seguenti attributi:

- Nome, attributo identificatore;
- Codice catastale;
- Provincia.

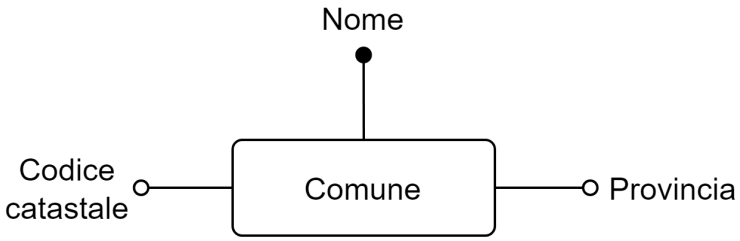

Figura 3.1: Entità Comune

Entità Ambulatorio

In Figura 3.2 è illustrata graficamente l'entità "Ambulatorio", che possiede i seguenti attributi:

- Nome, attributo identificatore;
- Giorno di apertura, attributo multivalore;
- Orario di apertura mattina;

- Orario di chiusura mattina;
- Orario di apertura pomeriggio;
- Orario di chiusura pomeriggio.

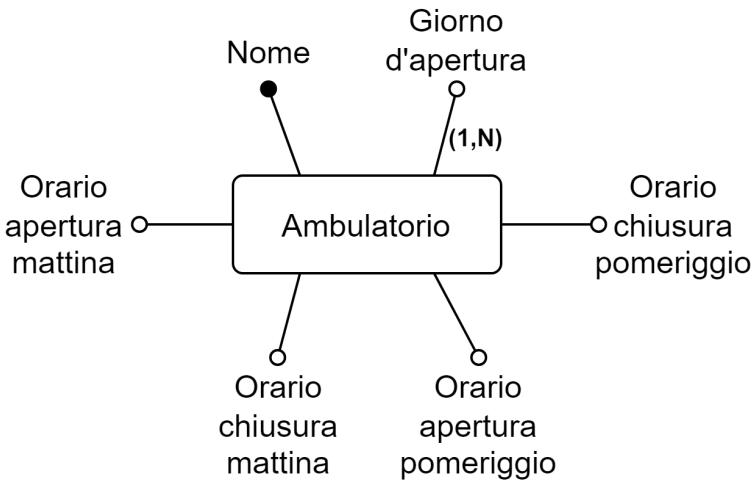

Figura 3.2: Entità Ambulatorio

Entità Specializzazione

In Figura 3.3 è illustrata graficamente l'entità "Specializzazione", che possiede i seguenti attributi:

- Nome, attributo identificatore.

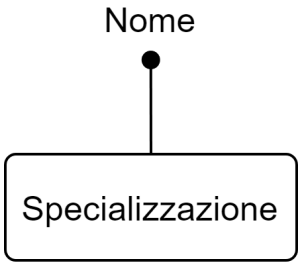

Figura 3.3: Entità Specializzazione

Entità Attività ambulatoriale

In Figura 3.4 è illustrata graficamente l'entità "Attività ambulatoriale", che possiede i seguenti attributi:

- Nome, attributo identificatore;
- Durata;
- Prezzo.

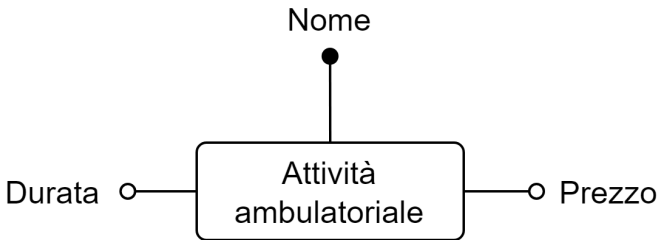

Figura 3.4: Entità Attività ambulatoriale

Entità Paziente

In Figura 3.5 è illustrata graficamente l'entità "Paziente", che possiede i seguenti attributi:

- ID paziente, attributo identificatore;
- Nome;
- Cognome;
- Codice fiscale;
- Sesso;
- Data di nascita, attributo composto che comprende:
  - Giorno di nascita;
  - Mese di nascita;
  - Anno di nascita.
- Città di nascita;
- Provincia di nascita;
- E-mail;
- Password.

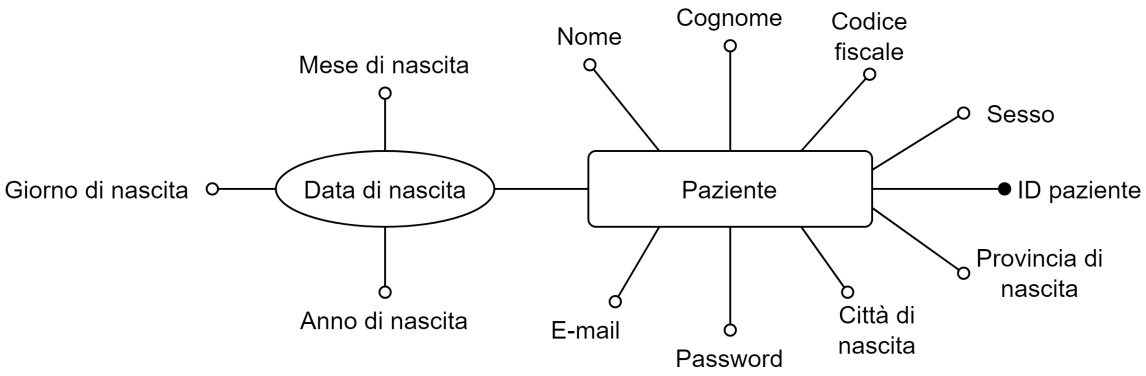

Figura 3.5: Entità Paziente

Entità Personale medico

In Figura 3.6 è illustrata graficamente l'entità "Personale medico", che possiede i seguenti attributi:

- ID personale medico, attributo identificatore;
- Nome;
- Cognome;
- Codice fiscale;
- Sesso;
- Data di nascita, attributo composto che comprende:
  - Giorno di nascita;
  - Mese di nascita;
  - Anno di nascita.
- Città di nascita;
- Provincia di nascita;
- Codice identificativo;
- E-mail;
- Password.

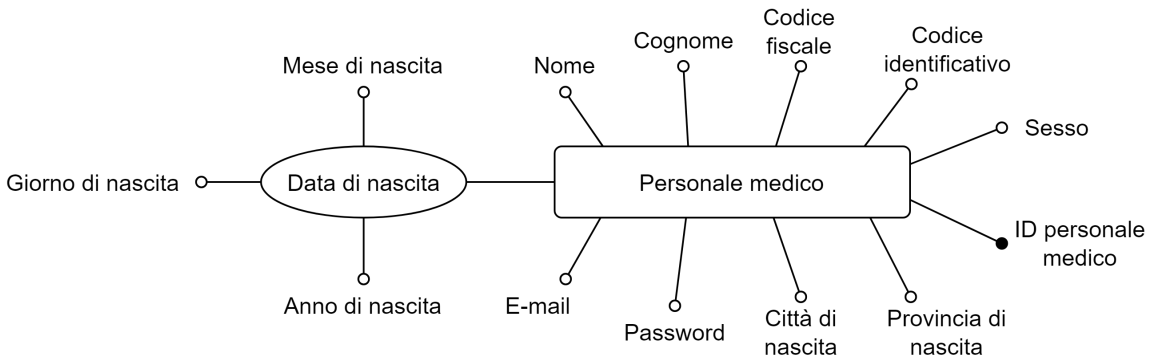

Figura 3.6: Entità Personale medico

Entità Prenotazione

In Figura 3.7 è illustrata graficamente l'entità "Prenotazione", che possiede i seguenti attributi:

- ID prenotazione, attributo identificatore;
- ID paziente;
- Nome paziente;
- Cognome paziente;
- ID personale medico;
- Nome personale medico;

- Cognome personale medico;
- Sesso personale medico;
- Ambulatorio;
- Attività ambulatoriale;
- Data e ora prenotazione;
- Data e ora visita;
- Durata visita;
- Prezzo visita;
- Descrizione patologia.

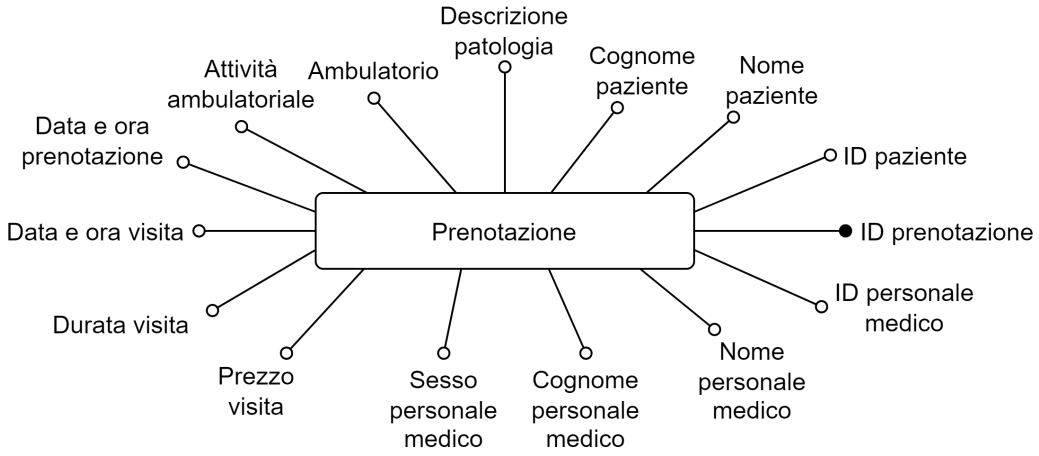

Figura 3.7: Entità Prenotazione

3.2.3 Identificazione degli attributi delle entità essenziali

Dall'analisi delle entità essenziali prodotte è stato possibile identificare le associazioni che legano le entità fondamentali e le generalizzazioni di entità, concetto che si basa sull'idea che alcune entità, condividendo attributi e caratteristiche simili, possono essere organizzate in una struttura gerarchica. In una gerarchia, un'entità "genitore" generalizza le caratteristiche comuni di entità più specifiche, note come "figlie". Le generalizzazioni possono essere suddivise in:

- *Generalizzazione totale*: ogni istanza dell'entità "genitore" deve essere associata almeno a un'entità "figlia".
- *Generalizzazione parziale*: un'istanza dell'entità "genitore" può non essere associata a un'entità "figlia", in quanto non possiedono caratteristiche comuni.

Generalizzazione totale delle entità Paziente e Personale medico

Poiché le entità "Paziente" e "Personale medico" posseggono molti attributi in comune, si ritiene opportuno aggiungere un'entità genitore chiamata "Utente", come illustrato in Figura 3.8.

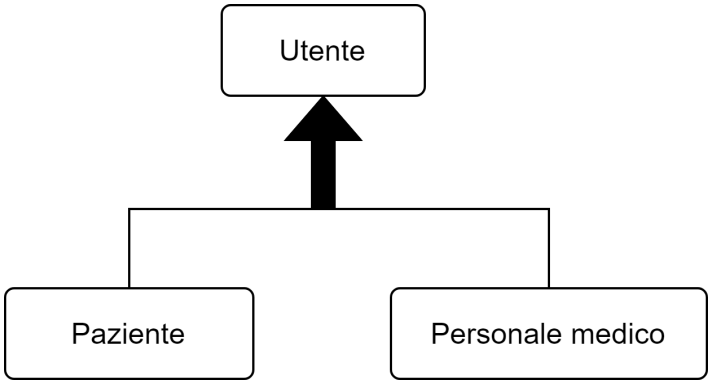

Figura 3.8: Generalizzazione Utente-Paziente-Personale medico

Associazione Nascita

Tra le entità "Utente" e "Comune" è possibile identificare l'associazione "Nascita", ovvero una relazione che indica per ogni utente qual è il relativo comune di nascita, come illustrato in Figura 3.9.

Figura 3.9: Associazione Nascita

Associazione Suddivisione ambulatorio

Tra le entità "Ambulatorio" e "Specializzazione" è possibile identificare l'associazione "Suddivisione ambulatorio", ovvero una relazione che associa ogni specializzazione a un unico ambulatorio, come illustrato in Figura 3.10.

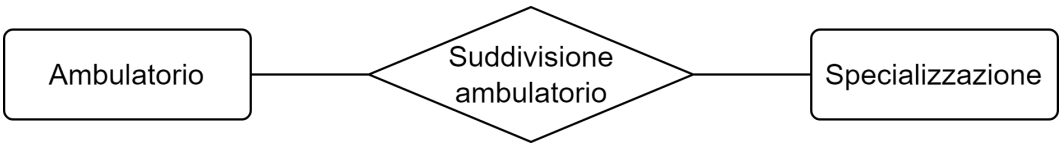

Figura 3.10: Associazione Suddivisione ambulatorio

Associazione Suddivisione specializzazione

Tra le entità "Specializzazione" e "Attività ambulatoriale" è possibile identificare l'associazione "Suddivisione specializzazione", ovvero una relazione che associa ogni attività ambulatoriale a un'unica specializzazione, come illustrato in Figura 3.11.

Figura 3.11: Associazione Suddivisione specializzazione

Associazione Visita

Tra le entità "Prenotazione", "Paziente", "Personale medico" e "Attività ambulatoriale" è possibile identificare l'associazione "Suddivisione specializzazione", ovvero una relazione che rappresenta la visita di un paziente presso la clinica, come illustrato in Figura 3.12. Da un'istanza di questa associazione possiamo conoscere chi è il paziente che deve essere visitato, la prenotazione effettuata dallo stesso, il personale medico che visiterà il paziente e l'attività ambulatoriale che egli ha scelto in fase di prenotazione.

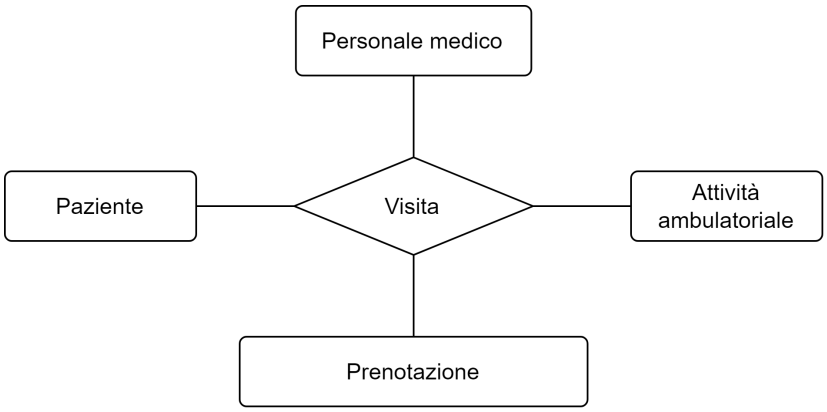

Figura 3.12: Associazione Visita

Altro

Per prevenire la formazione di cicli all'interno del modello, si è scelto di non stabilire una relazione diretta tra le entità "Personale medico" e "Ambulatorio". Al fine di evitare questa situazione, si è optato per includere l'attributo "Ambulatorio" all'interno della lista degli attributi dell'entità "Personale medico", creando, così, una ridondanza.

3.2.4 Modellazione dello schema concettuale scheletro

Dopo aver identificato le entità e le relazioni tra di esse, ovvero dopo aver individuato l'insieme degli elementi alla base dello schema concettuale che si intende progettare, l'applicazione delle tecniche di integrazione incluse nella strategia Bottom-Up ha reso possibile la modellazione di uno schema concettuale scheletro, dove per "scheletro" si intende l'assenza di attributi. Lo schema è stato progettato con l'obiettivo di offrire una visione completa delle interconnessioni tra le diverse entità. Lo schema concettuale scheletro è illustrato nella Figura 3.13.

3.2.5 Modifiche alla struttura delle entità

Con l'implementazione delle relazioni tra le diverse entità e l'introduzione dell'entità "Utente", che ha permesso di istituire una generalizzazione totale tramite le entità figlie "Paziente" e "Personale medico", la struttura di alcune entità è cambiata notevolmente.

Entità Utente

L'entità "Utente", illustrata in Figura 3.14, contiene l'insieme degli attributi condivisi tra le entità "Paziente" e "Personale medico". Questa entità presenta i seguenti attributi:

- Codice fiscale, attributo identificatore;

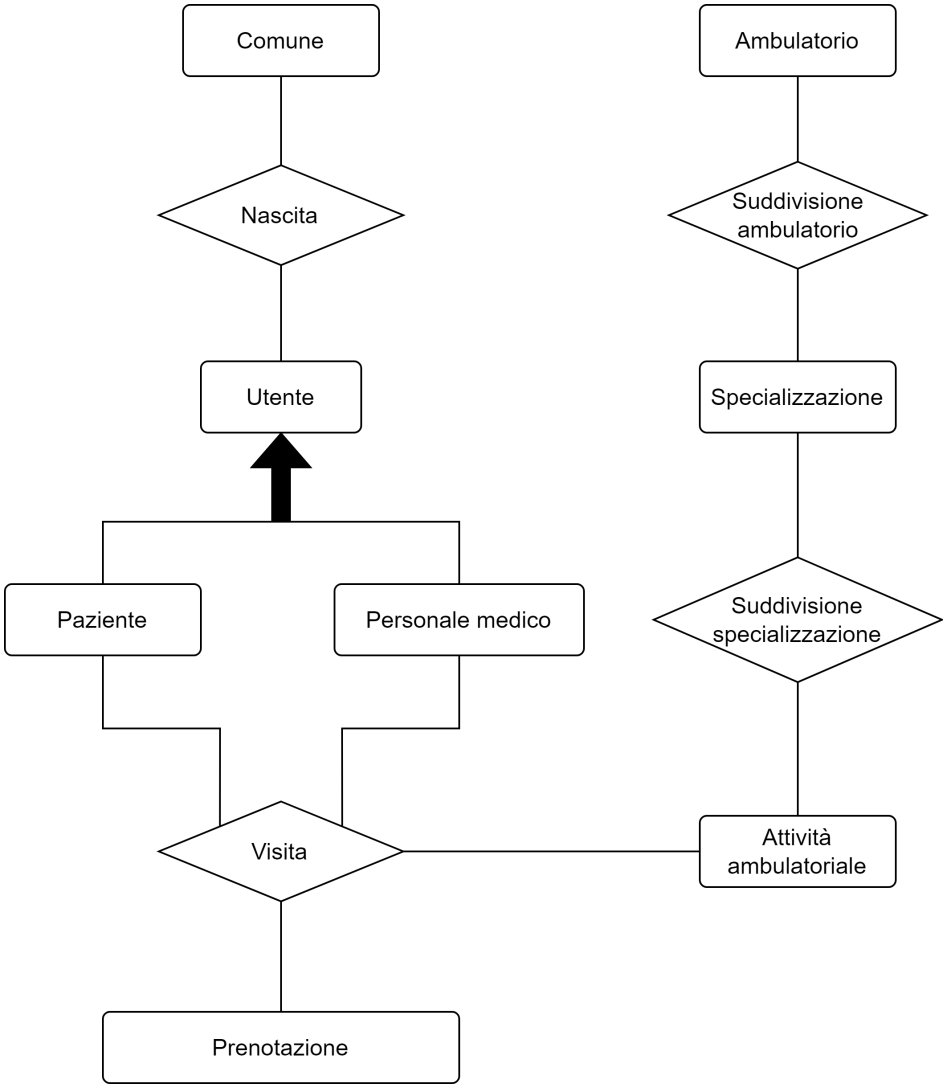

Figura 3.13: Schema concettuale scheletro

- Nome;
- Cognome;
- Sesso;
- Data di nascita, attributo composto che comprende:
  - Giorno di nascita;
  - Mese di nascita;
  - Anno di nascita.
- E-mail;
- Password.

Gli attributi "Città di nascita" e "Provincia di nascita" non fanno più parte della lista degli attributi di un utente, grazie alla presenza dell'associazione "Nascita".

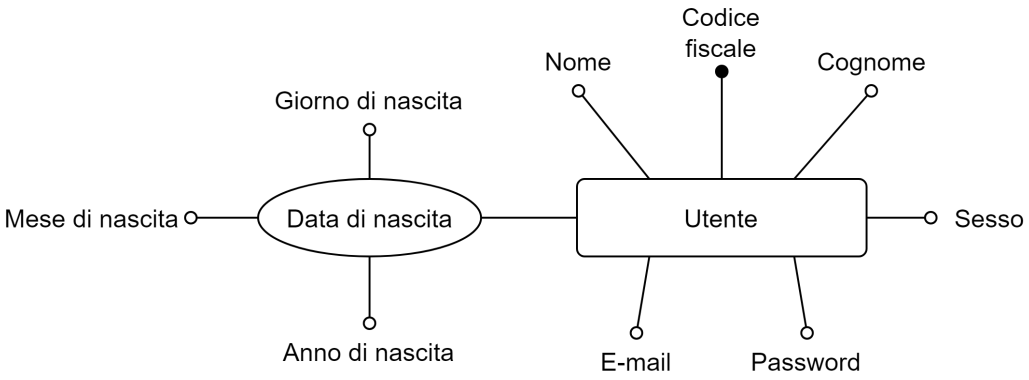

Figura 3.14: Entità Utente modificata

Entità Paziente

La maggior parte degli attributi dell'entità "Paziente" viene rimossa, in quanto presente nell'entità genitore della generalizzazione, ossia "Utente". L'unico attributo mantenuto è "ID paziente". La nuova rappresentazione dell'entità "Paziente" è illustrata in Figura 3.15.

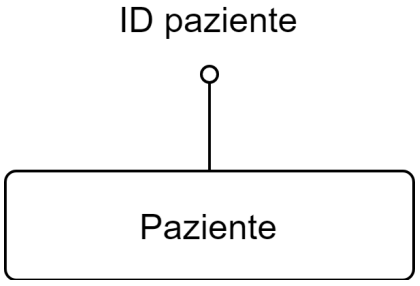

Figura 3.15: Entità Paziente modificata

Entità Personale medico

L'entità "Personale medico" segue un processo analogo a quello dell'entità "Paziente". La maggior parte degli attributi viene eliminata, mentre vengono mantenuti gli attributi "ID personale medico", "Codice identificativo" e "Ambulatorio". La nuova rappresentazione dell'entità "Personale medico" è illustrata in Figura 3.16.

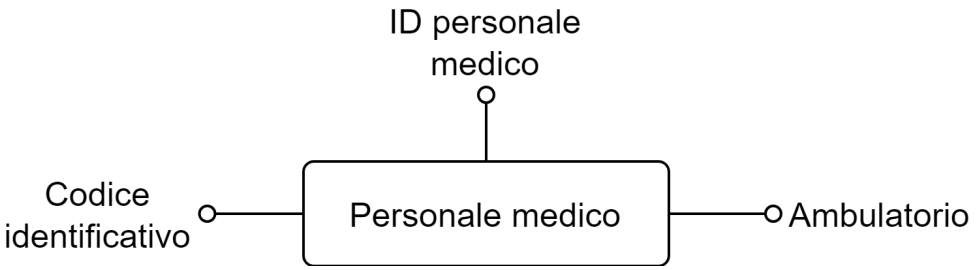

Figura 3.16: Entità Personale medico modificata

Entità Prenotazione

La maggior parte degli attributi dell'entità "Prenotazione" viene eliminata, poiché essa è collegata alle entità "Paziente", "Personale medico" e "Attività ambulatoriale" tramite la relazione "Visita". Gli attributi mantenuti sono i seguenti:

- ID prenotazione, attributo identificatore;
- Data e ora della prenotazione;
- Data e ora della visita;
- Descrizione della patologia.

La nuova rappresentazione dell'entità "Prenotazione" è illustrata in Figura 3.17.

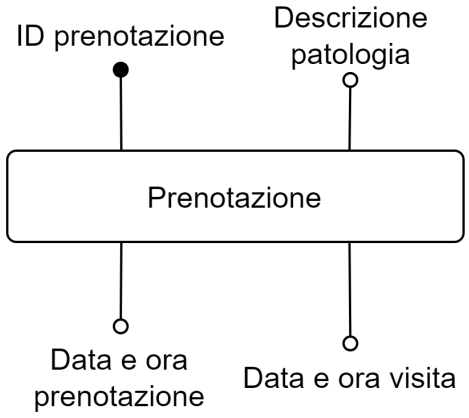

Figura 3.17: Entità prenotazione modificata

3.2.6 Modellazione dello schema concettuale completo

La modellazione dello schema concettuale non si limita alla semplice definizione delle entità e delle relazioni tra di esse, ma include anche il modo in cui le varie entità partecipano a tali relazioni, attraverso l'introduzione del concetto di cardinalità.

La cardinalità è solitamente espressa tramite coppie di numeri, dove il primo e il secondo numero indicano, rispettivamente, il numero minimo e massimo di istanze di un'entità che possono partecipare alla relazione. In altre parole, la cardinalità fornisce informazioni sulla quantità di partecipazione delle entità coinvolte nelle relazioni.

Nel nostro caso vengono utilizzate le coppie di numeri:

- (1,1): ogni istanza dell'entità A è associata esattamente a un'istanza dell'entità B, e viceversa.
- (0,N): ogni istanza dell'entità A può essere associata a nessuna o a molte istanze dell'entità B.
- (1,N): ogni istanza dell'entità A è associata ad almeno un'istanza dell'entità B e può essere associata a molte di esse.

L'intero schema E-R è rappresentato nella Figura 3.18.

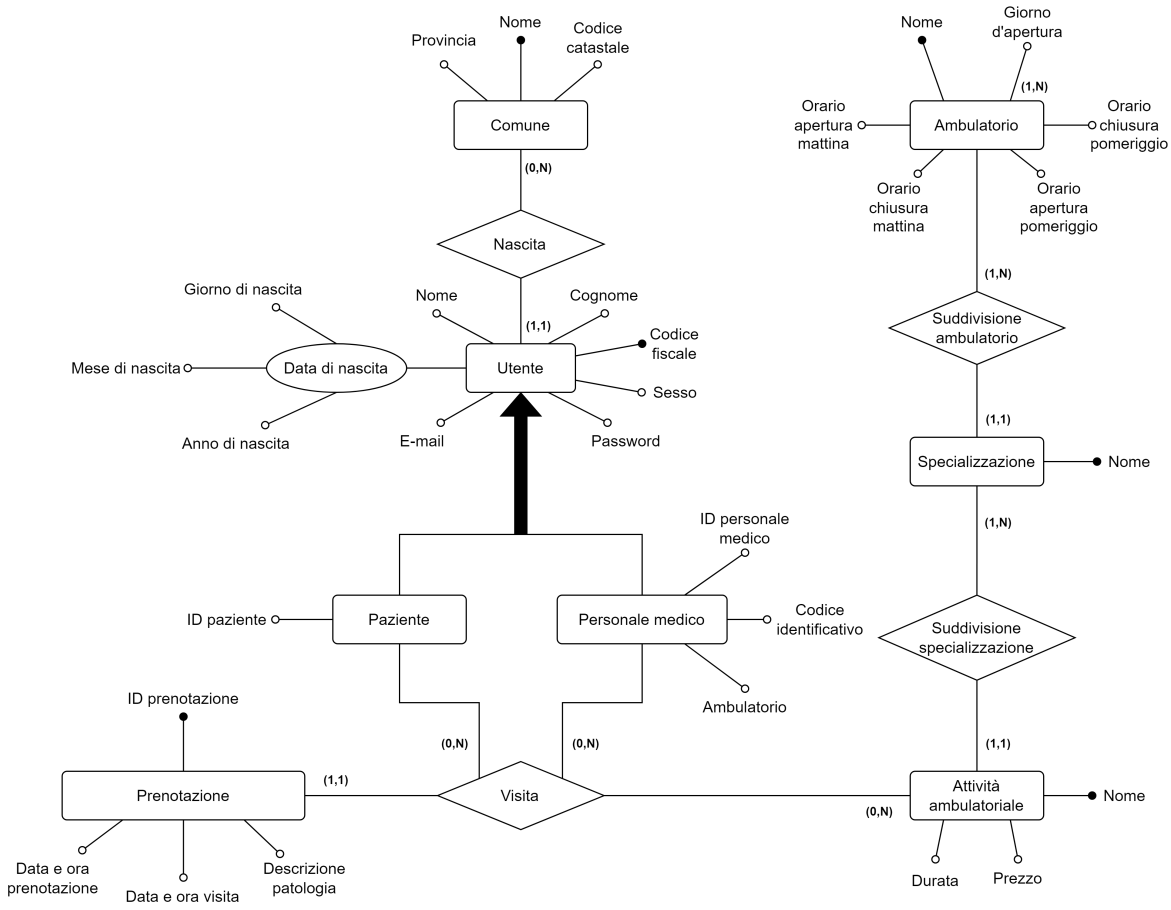

Figura 3.18: Schema concettuale completo

3.2.7 Vincoli non esprimibili

All'interno dello schema E-R ci sono alcuni vincoli che non possono essere pienamente espressi, ma che sono, comunque, fondamentali per definire il comportamento desiderato del sistema. Questi vincoli, che vanno oltre la semplice struttura delle entità e delle relazioni, si dividono in due principali categorie:

- *Vincoli intrarelazionali*: si riferiscono alle restrizioni imposte all'interno di una singola relazione o entità; essi possono essere utilizzati su singoli attributi o su gruppi di attributi.
- *Vincoli interrelazionali*: si riferiscono alle restrizioni imposte tra diverse relazioni o entità, preservando, quindi, la coerenza dei dati.

Di seguito vengono elencati i vincoli che, sebbene non siano stati rappresentati tramite i costrutti del modello concettuale, rivestono un'importanza significativa all'interno del modello progettato.

Vincoli intrarelazionali

1. L'attributo "Codice catastale" relativo all'entità "Comune" deve essere una stringa alfanumerica, composta da un carattere alfabetico seguito da tre cifre, ad esempio: "D643".

2. L'attributo "Codice fiscale" relativo all'entità "Cittadino" deve essere una stringa alfanumerica di 16 caratteri.
3. L'attributo "Sesso" relativo all'entità "Utente" ha come valori ammissibili 'M' o 'F'; qualunque altro carattere o stringa sono da ritenersi errati.
4. L'attributo "E-mail" relativo all'entità "Utente" deve essere una stringa che segua il formato di un indirizzo e-mail valido. La stringa deve contenere almeno un simbolo "@" e almeno un punto a seguire, in modo da rappresentare la struttura tipica di un indirizzo e-mail. Qualsiasi valore che non rispetti questo formato è considerato non valido e non sarà accettato.
5. L'attributo "Password" relativo all'entità "Utente" deve essere una stringa che rispetti le seguenti regole di sicurezza:
   - essere lunga almeno otto caratteri;
   - contenere almeno una lettera maiuscola;
   - contenere almeno una lettera minuscola;
   - contenere almeno un numero;
   - contenere almeno un carattere speciale come !, $, # e .
6. L'attributo "ID paziente" relativo all'entità "Paziente" deve assumere valori numerici progressivi a partire dal numero 1.
7. L'attributo "ID personale medico" relativo all'entità "Personale medico" deve assumere valori numerici progressivi a partire dal numero 1.
8. L'attributo "Codice identificativo" relativo all'entità "Personale medico" deve seguire le seguenti regole:
   - iniziare con la stringa "CPCA" che rappresenta l'acronimo di Clinica Privata Casa Alfredo;
   - proseguire con una sigla rappresentante l'ambulatorio di riferimento (ad esempio, "MGC" per Medicina Generale Cardiologica);
   - terminare con la stringa "pm" che denota "personale medico", seguita dall'ID univoco del personale medico.
9. L'attributo "ID prenotazione" relativo all'entità "Prenotazione" assume valori numerici progressivi a partire dal numero 1.

Vincoli interrelazionali

Per quanto riguarda i vincoli interrelazionali, è stato individuato un unico vincolo che stabilisce quanto segue: l'attributo "Ambulatorio" relativo all'entità "Personale medico" coincide con l'attributo "Nome" dell'entità "Ambulatorio".

3.3 Dizionario dei dati

Il dizionario dei dati è un documento che fornisce un'analisi esaustiva delle informazioni presenti nel diagramma E-R, descrivendo in modo dettagliato le entità e le relazioni dello schema. Esso, pertanto, consente una comprensione accurata delle entità coinvolte e delle loro interazioni.

3.3.1 Entità

Nella Tabella 3.1 viene illustrato il dizionario dei dati delle entità, che descrive tutte le entità presenti nello schema E-R, dando, inoltre, una definizione del tipo degli attributi associati e degli identificatori relativi.

Nome entità	Descrizione	Attributi	Identificatore
Comune	Unità amministrativa e territoriale che rappresenta una suddivisione del territorio di uno Stato. Nel nostro caso le istanze di questa entità si riferiscono solo a comuni italiani	• Nome (stringa) • Provincia (stringa) • **Codice catastale** (stringa) "**Codice catastale**" indica il codice utilizzato dall'amministrazione italiana per identificare un comune italiano	Nome
Utente	Entità che usufruisce del sistema	• Codice fiscale (stringa) • Nome (stringa) • Cognome (stringa) • Sesso (carattere) • Giorno di nascita (numerico) • Mese di nascita (numerico) • Anno di nascita (numerico) • E-mail (stringa) • Password (stringa)	Codice fiscale
Paziente	Chi è affidato alle cure di un medico	• Tutti gli attributi dell'entità genitore "Utente" • ID paziente (numerico)	Lo stesso identificatore dell'entità genitore "Utente"
Personale medico	Chi opera all'interno della clinica	• Tutti gli attributi dell'entità genitore "Utente" • ID personale medico (numerico) • Codice identificatico (stringa) • **Ambulatorio** (stringa) "**Ambulatorio**" indica il reparto in cui il personale medico opera	Lo stesso identificatore dell'entità genitore "Utente"
Ambulatorio	Locale attrezzato per visite mediche e cure specialistiche che non richiedano una degenza ospedaliera	• Nome (stringa) • Giorno di apertura (stringa) • Orario di apertura mattina (numerico) • Orario di chiusura mattina (numerico) • Orario di apertura pomeriggio (numerico) • Orario di chiusura pomeriggio (numerico)	Nome

Specializzazione	Ambiti in cui si può suddividere un ambulatorio	• Nome (stringa)	Nome
Attività ambulatoriale	Atto clinico-assistenziale, di natura diagnostica e/o terapeutica, erogato da professionisti/strutture nei riguardi di un paziente	• Nome (stringa) • **Durata** (numerico) • **Prezzo** (numerico) "**Durata**" indica la durata in media dell'attività ambulatoriale. "**Prezzo**" indica il prezzo dell'attività ambulatoriale in euro.	Nome
Prenotazione	Azione che permette a un Paziente di effettuare una visita all'interno della clinica	• ID prenotazione (numerico) • Data e ora prenotazione (datetime) • Data e ora visita (datetime) • **Descrizione patologia** (stringa) "**Descrizione patologia**" assume un valore solo se la visita è già stata effettuata e il personale medico che ha visitato il paziente ne ha riempito il campo, altrimenti l'attributo assume valore nullo.	ID prenotazione

Tabella 3.1: Dizionario dei dati delle entità

3.3.2 Relazioni

Nella Tabella 3.2 viene illustrato il dizionario dei dati delle relazioni; in esso vengono descritte tutte le associazioni presenti nello schema E-R elencando le entità coinvolte in ognuna di esse, con le rispettive cardinalità.

Nome relazione	Descrizione	Entità coinvolte
Nascita	Associa un utente a un comune di nascita	• Utente (1,1) • Comune (0,N)
Suddivisione ambulatorio	Associa a ciascuna specializzazione il relativo ambulatorio	• Ambulatorio (1,N) • Specializzazione (1,1)
Suddivisione specializzazione	Associa a ciascuna attività ambulatoriale la relativa specializzazione	• Specializzazione (1,N) • Attività ambulatoriale (1,1)
Visita	Associa a ciascuna Prenotazione un Paziente, un Personale medico e un'Attività ambulatoriale	• Prenotazione (1,1) • Paziente (0,N) • Personale medico (0,N) • Attività ambulatoriale (0,N)

Tabella 3.2: Dizionario dei dati delle relazioni

3.4 Progettazione logica

La progettazione logica delle basi di dati è una fase intermedia tra la progettazione concettuale (modello E-R) e la creazione del modello logico. Questa fase si concentra sulla trasformazione del modello concettuale, precedentemente illustrato, in uno schema di database dettagliato e implementabile. Alcuni dei passi principali di questa fase, che descriveremo nel corso del paragrafo, includono:

- *Eliminazione delle generalizzazioni*. Durante questa fase, si decide come gestire le gerarchie di generalizzazione presenti nel modello concettuale. Si può optare per il collasso verso l'alto (si riuniscono gli attributi delle entità figlie nell'entità padre, aggiungendo un attributo di tipo), il collasso verso il basso (si elimina l'entità padre trasferendone gli attributi su tutte le entità figlie) o il mantenimento sia dell'entità genitore che delle entità figlie con modifiche minime.
- *Eliminazione di attributi composti*. Gli attributi composti vengono suddivisi in attributi atomici; ciò facilita la gestione dei dati e l'implementazione, poiché gli attributi sono più facilmente accessibili e modificabili.
- *Eliminazione di attributi multi-valore*. Gli attributi multi-valore, che possono avere più valori associati a una singola entità, vengono gestiti creando una nuova entità, in cui ogni istanza rappresenta un valore dell'attributo.

3.4.1 Ristrutturazione dello schema E-R

Durante la fase di ristrutturazione dello schema E-R verranno applicate le regole di progettazione logica precedentemente descritte, con l'obiettivo di tradurre il modello concettuale in uno schema di database adatto all'implementazione.

3.4.1.1 Eliminazione di generalizzazioni

L'obiettivo principale di questa fase è l'eliminazione della gerarchia nello schema E-R tra le entità "Utente", "Paziente" e "Personale medico". Si è deciso di conservare sia l'entità padre che le entità figlie, optando per la sostituzione della generalizzazione con due relazioni:

- "Dati Paziente", che associa le entità "Paziente" e "Utente";
- "Dati Personale medico", che associa le entità "Personale medico" e "Utente".

Eliminando la generalizzazione, le entità "Paziente" e "Personale medico" hanno come attributo identificatore, rispettivamente, "ID paziente" e "ID personale medico". Le modifiche effettuate in questa fase vengono illustrate in dettaglio nella Figura 3.19.

3.4.1.2 Eliminazione di attributi composti

Questa sezione è finalizzata ad eliminare l'attributo composto relativo all'entità "Utente". L'attributo in questione è "Data di nascita", come richiamato in Figura 3.20.

Si può procedere con due possibili approcci:

- viene sostituito l'attributo "Data di nascita" da quelli che esso comprende;
- viene utilizzato esclusivamente l'attributo "Data di nascita", che comprende, nel valore che assume, tutte le informazioni che sarebbero state contenute negli attributi "Giorno di nascita", "Mese di nascita" e "Anno di nascita".

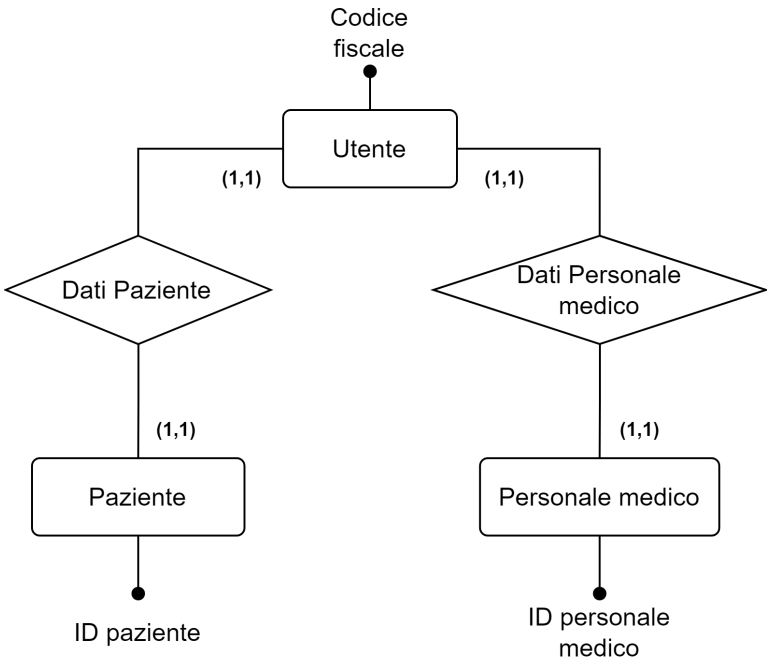

Figura 3.19: Eliminazione della generalizzazione tra Utente, Paziente e Personale medico

Figura 3.20: Attributo composto "Data di nascita"

Si è scelto di adottare l'ultima opzione, in quanto il tipo di dato implementabile per l'attributo "Data di nascita" è *DATE*, che consente di rappresentare una data completa, comprendente anno, mese e giorno, in modo compatto.

3.4.1.3 Eliminazione di attributi multi-valore

Al fine di eliminare l'attributo multi-valore "Giorni di apertura" nell'entità "Ambulatorio", si è stabilito di ristrutturare lo schema inserendo una nuova entità, cioè "Giorno", dalla quale dipende l'attributo "Nome" (suo identificatore) e la relativa associazione "Giorno di apertura"; quest'ultima mette in relazione l'entità "Giorno" con l'entità "Ambulatorio". Le modifiche proposte sono illustrate in Figura 3.21.

3.4.1.4 Schema E-R ristrutturato

In Figura 3.22 viene illustrato lo schema E-R ristrutturato, mediante l'applicazione delle fasi precedentemente descritte allo schema concettuale, ottenuto come prodotto finale della progettazione concettuale.

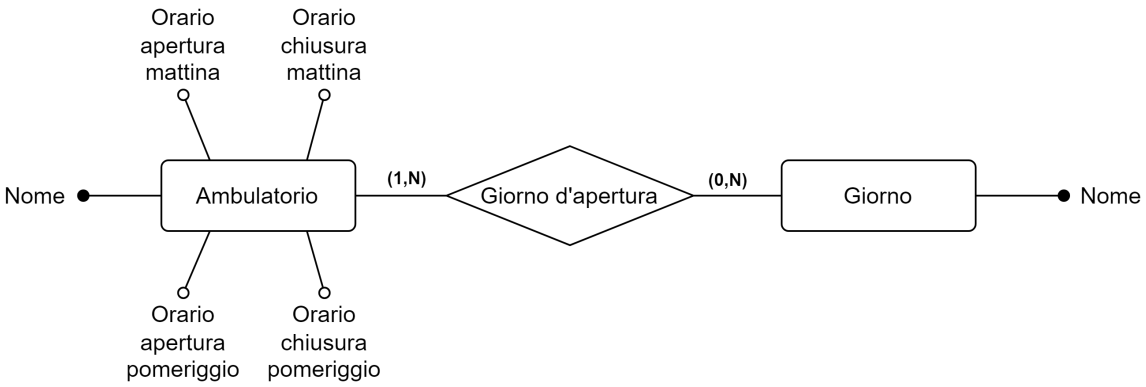

Figura 3.21: Eliminazione dell'attributo multivalore "Giorno d'apertura"

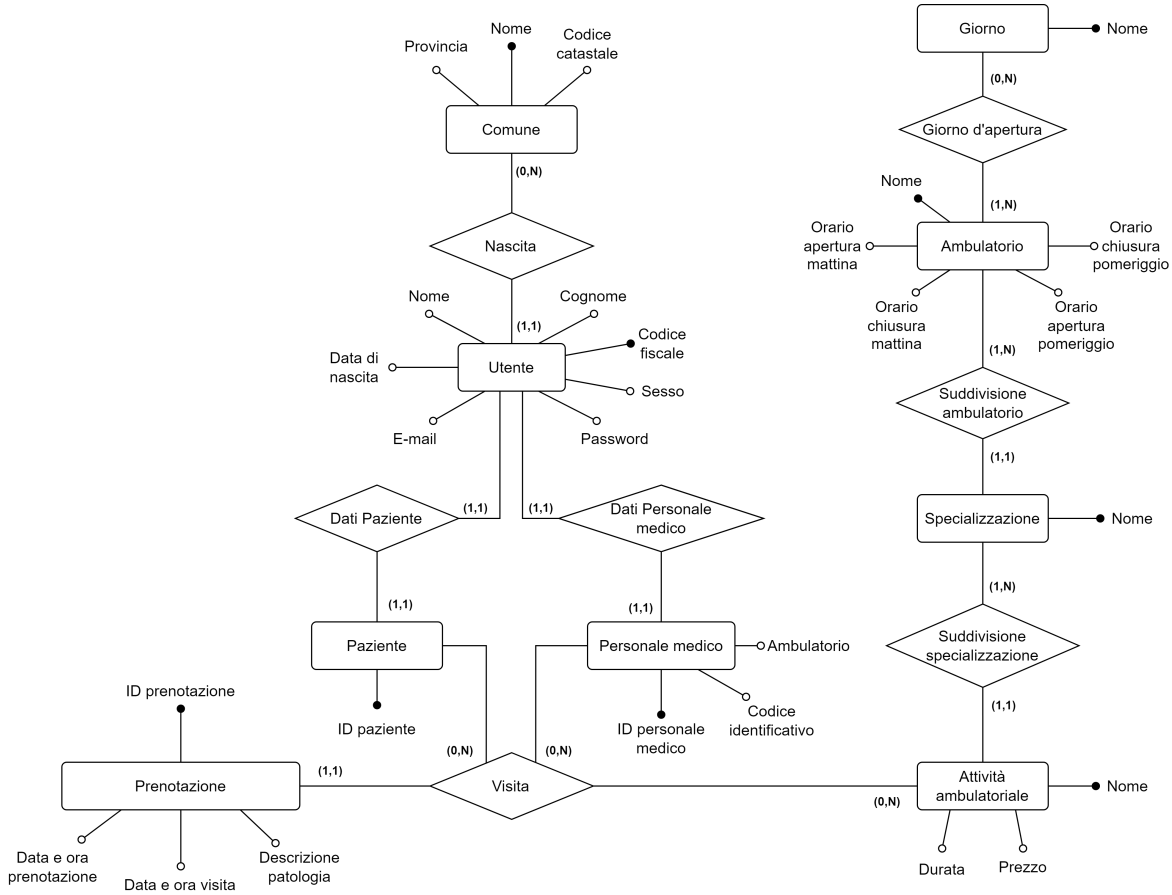

Figura 3.22: Schema completo e ristrutturato

3.4.2 Scelta degli identificatori primari

La scelta degli identificatori primari, o chiavi primarie, per le entità è una fase molto importante, che dà inizio alla traduzione dello schema E-R nel modello relazionale. Le chiavi primarie sono utilizzate per identificare univocamente ogni istanza di ogni entità del modello; la loro scelta deve essere accurata per garantire che i dati siano strutturati correttamente.

Nella Tabella 3.3 sono definiti gli identificatori primari per ciascuna entità. È importante notare che, avendo già effettuato la scelta degli identificatori nella fase di progettazione concettuale, gli identificatori primari sono rimasti invariati.

Nome Entità	Identificatore
Giorno	Nome
Ambulatorio	Nome
Specializzazione	Nome
Attività ambulatoriale	Nome
Comune	Nome
Utente	Codice fiscale
Paziente	ID paziente
Personale medico	ID personale medico
Prenotazione	ID prenotazione

Tabella 3.3: Identificatori primari delle entità

3.4.3 Modello relazionale

La traduzione dello schema E-R nel modello relazionale è una fase critica del processo di progettazione di un database. Durante questa fase, il modello concettuale, che utilizza entità, attributi e relazioni per rappresentare i dati in modo astratto, viene trasformato in un modello di database relazionale, che utilizza tabelle, colonne e chiavi per rappresentare i dati in una forma più concreta e organizzata.

Nella Tabella 3.4 viene mostrato lo schema relazionale, ottenuto dallo schema concettuale ristrutturato.

Legenda

- Attributo: indica la chiave primaria della relazione.
- Attributo*<Entità>: indica un vincolo di integrità referenziale tra la chiave di Entità e Attributo.

Entità/Relazione	Modello relazionale
Giorno	Giorno (Nome)
Ambulatorio	Ambulatorio (Nome, Orario di apertura mattina, Orario di chiusura mattina, Orario di apertura pomeriggio, Orario di chiusura pomeriggio)
Giorno di apertura	Giorno di apertura (Ambulatorio*<Ambulatorio>Giorno*<Giorno>)
Specializzazione	Specializzazione (Nome, Ambulatorio*<Ambulatorio>)
Attività ambulatoriale	Attività ambulatoriale (Nome, Durata, Prezzo, Specializzazione*<Specializzazione>)
Comune	Comune (Nome, Provincia, Codice catastale)
Utente	Utente (Codice fiscale, Nome, Cognome, Sesso, Data di nascita, E-mail, Password, Comune di nascita*<Comune>)
Paziente	Paziente (ID paziente, Dati personali*<Utente>)
Personale medico	Personale medico (ID personale medico, Codice identificativo, Ambulatorio, Dati personali*<Utente>)

Prenotazione	Prenotazione (ID prenotazione, Data e ora prenotazione, Data e ora visita, Descrizione patologia, Paziente*<Paziente>, Personale medico*<Personale medico>, Attività ambulatoriale*<Attività ambulatoriale>)

Tabella 3.4: Schema relazionale del database

3.4.4 Definizione delle tabelle

Nella fase di definizione delle tabelle vengono create e definite le tabelle che rappresenteranno i dati nel database. Il DDL (Data Definition Language), che è parte del linguaggio SQL, verrà utilizzato per eseguire comandi, come CREATE TABLE, per la creazione delle tabelle.

Durante questa fase, si definiscono le colonne di ciascuna tabella, specificando il nome della colonna, il tipo di dati e le restrizioni, come chiavi primarie e chiavi esterne.

Tabella comune

```
CREATE TABLE comune(
nome VARCHAR(255) PRIMARY KEY,
provincia VARCHAR(255) NOT NULL,
codice_catastale CHAR(4) NOT NULL CHECK(
  codice_catastale REGEXP '^[A-Z][0-9]{3}$'
 )
)
```

Listato 3.1: Creazione della tabella `comune`

Tabella utente

```
CREATE TABLE utente(
codice_fiscale CHAR(16) PRIMARY KEY CHECK(
  codice_fiscale REGEXP '^[A-Z0-9]{16}$'
 ),
nome VARCHAR(255) NOT NULL,
cognome VARCHAR(255) NOT NULL,
sesso CHAR(1) NOT NULL CHECK(sesso IN('M','F')),
data_di_nascita DATE NOT NULL,
email VARCHAR(255) NOT NULL CHECK(
  email REGEXP '^[A-Za-z0-9\.\-\_]+@([A-Za-z\-]+\.)+[A-Z-a-z\-]{2,4}$'
 ),
password VARCHAR(255) NOT NULL CHECK(
  password REGEXP '^(?=.*[0-9])(?=.*[a-z])(?=.*[A-Z])
  (?=.*[*.!@$\%^&()#{}[\]:;<>,.?\/~_+\-=|]).{8,32}$'
 ),
comune_di_nascita VARCHAR(255) NOT NULL,
FOREIGN KEY(comune_di_nascita) REFERENCES comune(nome)
ON UPDATE CASCADE ON DELETE NO ACTION
)
```

Listato 3.2: Creazione della tabella `utente`

Tabella paziente

```
CREATE TABLE paziente(
id_paziente SMALLINT PRIMARY KEY AUTO_INCREMENT,
dati_personali CHAR(16) NOT NULL,
FOREIGN KEY(dati_personali) REFERENCES utente(codice_fiscale)
ON UPDATE CASCADE ON DELETE NO ACTION
)
```

Listato 3.3: Creazione della tabella `paziente`

Tabella personale_medico

```
CREATE TABLE personale_medico(
id_personale_medico SMALLINT PRIMARY KEY AUTO_INCREMENT,
codice_identificativo VARCHAR(14) NOT NULL CHECK(
  codice_identificativo REGEXP '^CPCA[A-Z]{3}pm[0-9]{1,5}$'
 ),
ambulatorio VARCHAR(30) NOT NULL,
dati_personali CHAR(16) NOT NULL,
FOREIGN KEY(dati_personali) REFERENCES utente(codice_fiscale)
ON UPDATE CASCADE ON DELETE NO ACTION
)
```

Listato 3.4: Creazione della tabella `personale_medico`

Tabella giorno

```
CREATE TABLE giorno(
nome VARCHAR(8) PRIMARY KEY
)
```

Listato 3.5: Creazione della tabella `giorno`

Tabella ambulatorio

```
CREATE TABLE ambulatorio(
Nome VARCHAR(30) PRIMARY KEY,
orario_apertura_mattina CHAR(2) NOT NULL,
orario_chiusura_mattina CHAR(2) NOT NULL,
orario_apertura_pomeriggio CHAR(2) NOT NULL,
orario_chiusura_pomeriggio CHAR(2) NOT NULL
)
```

Listato 3.6: Creazione della tabella `ambulatorio`

Tabella giorno_di_apertura

```
CREATE TABLE giorno_di_apertura(
ambulatorio VARCHAR(30) NOT NULL,
giorno VARCHAR(9) NOT NULL,
PRIMARY KEY(ambulatorio, giorno),
FOREIGN KEY(ambulatorio) REFERENCES ambulatorio(nome)
ON UPDATE CASCADE ON DELETE NO ACTION,
FOREIGN KEY(giorno) REFERENCES giorno(nome)
ON UPDATE CASCADE ON DELETE NO ACTION
)
```

Listato 3.7: Creazione della tabella `giorno_di_apertura`

Tabella specializzazione

```
CREATE TABLE specializzazione(
nome VARCHAR(255) PRIMARY KEY,
ambulatorio VARCHAR(30) NOT NULL,
FOREIGN KEY(ambulatorio) REFERENCES ambulatorio(nome)
ON UPDATE CASCADE ON DELETE NO ACTION
)
```

Listato 3.8: Creazione della tabella `specializzazione`

Tabella attivita_ambulatoriale

```
CREATE TABLE attivita_ambulatoriale(
nome VARCHAR(255) PRIMARY KEY,
durata SMALLINT NOT NULL,
prezzo VARCHAR(4) NOT NULL,
specializzazione VARCHAR(255) NOT NULL,
FOREIGN KEY(specializzazione) REFERENCES specializzazione(nome)
ON UPDATE CASCADE ON DELETE NO ACTION
)
```

Listato 3.9: Creazione della tabella `attivita_ambulatoriale`

Tabella prenotazione

```
CREATE TABLE prenotazione(
id_prenotazione INTEGER PRIMARY KEY AUTO_INCREMENT,
data_ora_prenotazione DATETIME NOT NULL,
data_ora_visita DATETIME NOT NULL,
descrizione_patologia VARCHAR(255),
paziente SMALLINT NOT NULL,
personale_medico SMALLINT NOT NULL,
attivita_ambulatoriale VARCHAR(255) NOT NULL,
FOREIGN KEY(paziente) REFERENCES paziente(id_paziente)
```

```
10 ON UPDATE CASCADE ON DELETE NO ACTION,
11 FOREIGN KEY(personale_medico) REFERENCES personale_medico(id_personale_medico
12 ON UPDATE CASCADE ON DELETE NO ACTION,
13 FOREIGN KEY(attivita_ambulatoriale) REFERENCES attivita_ambulatoriale(nome)
14 ON UPDATE CASCADE ON DELETE NO ACTION
15 )
```

Listato 3.10: Creazione della tabella `prenotazione`

CAPITOLO 4

Progettazione delle applicazioni

In questo capitolo affronteremo la progettazione delle applicazioni per la clinica privata "Casa Alfredo", in cui impiegheremo il linguaggio di modellazione UML. In particolare, utilizzeremo i diagrammi delle classi e delle attività. La fase di progettazione è molto importante nello sviluppo di un prodotto software, in quanto può evitare, durante l'implementazione, l'incremento di problematiche legate all'architettura del software.

4.1 Scelta del pattern architetturale

Nella progettazione delle applicazioni software per la clinica privata "Casa Alfredo" la scelta del pattern architetturale rappresenta un passaggio fondamentale per garantire l'efficienza, la manutenibilità e la scalabilità del sistema. Tra le opzioni disponibili, si è optato per il pattern MVCS (Model-View-Controller-Service), che è una evoluzione sofisticata del ben noto pattern MVC (Model-View-Controller), comunemente utilizzato nella progettazione del software. A differenza di quest'ultimo, il pattern MVCS aggiunge un quarto strato, tra "Controller" e "Model", denominato "Service", che si interfaccia con il database. Con il pattern MVCS l'applicazione viene suddivisa in quattro componenti interconnessi, ciascuno con un ruolo specifico nel sistema.

- *Model*: si concentra principalmente sulla rappresentazione dei dati e sul mantenimento della loro coerenza all'interno dell'applicazione.
- *View*: è responsabile della presentazione dei dati all'utente. Essa visualizza i dati estratti dal Model in un formato comprensibile e fornisce un'interfaccia per l'input e l'output. La View non contiene logica di business rilevante, limitandosi a inoltrare le richieste al Controller.
- *Controller*: gestisce le interazioni tra l'utente e l'applicazione. Accetta le richieste dell'utente attraverso la View, le processa e le inoltra al Service per ottenere i Model appropriati. Pur eseguendo operazioni di validazione e gestendo la logica di business, il Controller si concentra principalmente sul controllo del flusso dell'applicazione e sull'aggiornamento della View in risposta alle richieste inviate al Service.
- *Service*: fornisce una serie di funzioni che il Controller può utilizzare per accedere ai dati del Model. Il Service comunica con il database per recuperare, inserire, aggiornare o eliminare dati. Questa comunicazione con il database avviene attraverso query SQL o

mediante l'utilizzo di un ORM (Object-Relational Mapping), se l'applicazione ne fa uso. Il Service si occupa di creare e gestire le query necessarie per interagire con il database.

In sintesi, il pattern MVCS è una estensione avanzata dell'MVC, progettata per affrontare applicazioni più complesse, data la presenza di un database, favorendo la separazione delle responsabilità e contribuendo a una manutenzione agevole del codice. La suddivisione dei componenti in quattro entità interconnesse assicura chiarezza nell'architettura e facilità nell'implementazione di applicazioni robuste.

4.2 Diagrammi UML

I diagrammi UML sono una notazione grafica standard utilizzata per modellare e documentare il design di sistemi software. Questi diagrammi forniscono una visualizzazione chiara e concettuale di come un sistema è organizzato e di come interagiscono le sue parti. Ci sono diversi tipi di diagrammi UML, ciascuno dei quali si concentra su un aspetto specifico del sistema. Noi descriveremo il sistema della clinica privata "Casa Alfredo" utilizzando il "Diagramma delle classi" e il "Diagramma delle attività".

4.2.1 Diagramma delle classi

Il diagramma delle classi è uno dei diagrammi UML più utilizzati e viene impiegato per rappresentare la struttura statica di un sistema. Esso prevede i seguenti costrutti:

- *Classi*. Le classi sono rappresentate da un rettangolo diviso in tre sezioni. La parte superiore contiene il nome della classe; la sezione centrale elenca gli attributi della classe (variabili di istanza o campi) insieme ai loro tipi di dati; la sezione inferiore contiene i metodi della classe (funzioni o operazioni) con i loro parametri e tipi di ritorno.
- *Costruttore*. Un costruttore è un metodo speciale utilizzato per creare nuovi oggetti della classe. Di solito ha lo stesso nome della classe e può avere parametri per inizializzare gli attributi. Esso viene elencato assieme ai metodi di una classe.
- *Associazioni*. Le associazioni rappresentano le relazioni tra le classi. Possono essere unidirezionali o bidirezionali e possono avere cardinalità che indicano quanti oggetti di una classe sono associati agli oggetti di un'altra classe.

Una volta definite, le classi possono essere organizzate in gruppi logici, chiamati "package", rappresentati come cartelle contenenti un insieme di classi correlate. Questa disposizione aiuta a organizzare in modo più chiaro ed efficiente grandi sistemi software.

Inizieremo descrivendo dettagliatamente ciascun package; successivamente, verrà presentato uno schema generale per visualizzare i collegamenti tra le diverse classi.

4.2.1.1 Package Utente

In Figura 4.1 è illustrato il package "Utente" che contiene una sola classe, di tipo View. Quest'ultima rappresenta la schermata iniziale, che viene mostrata quando l'utente avvia l'applicazione.

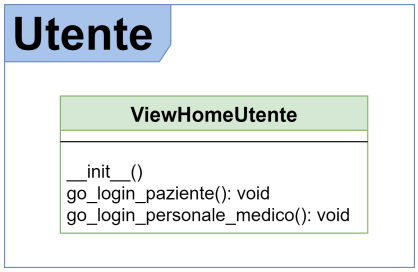

Figura 4.1: Package Utente

4.2.1.2 Package Paziente

In Figura 4.2 è illustrato il package "Paziente" che raggruppa i sotto-package "Login Paziente", "View" e "Controller". La Figura descrive come le singole classi dei diversi package interagiscono tra loro.

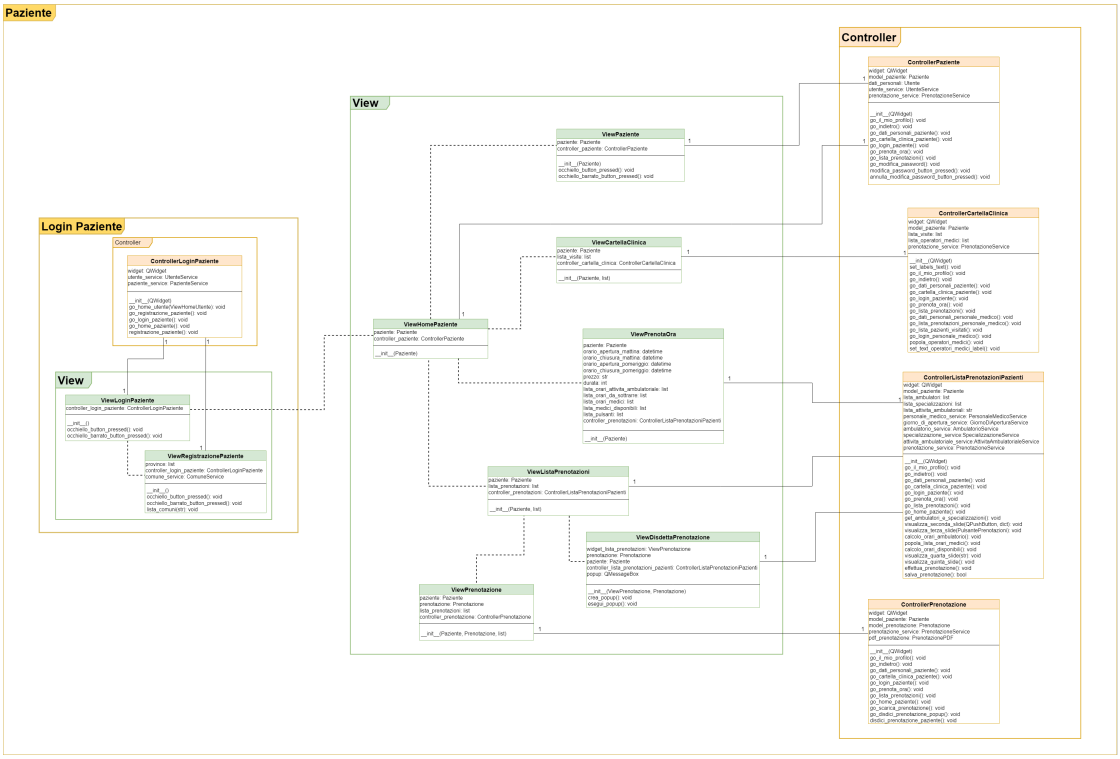

Figura 4.2: Package Paziente

Descriviamo, ora, i singoli sotto-package.

Sotto-package Login Paziente

In Figura 4.3 è illustrato il sotto-package "Login Paziente". Esso raggruppa le View e il Controller che si occupano dell'autenticazione e della registrazione del paziente.

Sotto-package View

In Figura 4.4 è illustrato il sotto-package "View", il quale raggruppa le classi di tipo View che possono essere mostrate a un paziente autenticato.

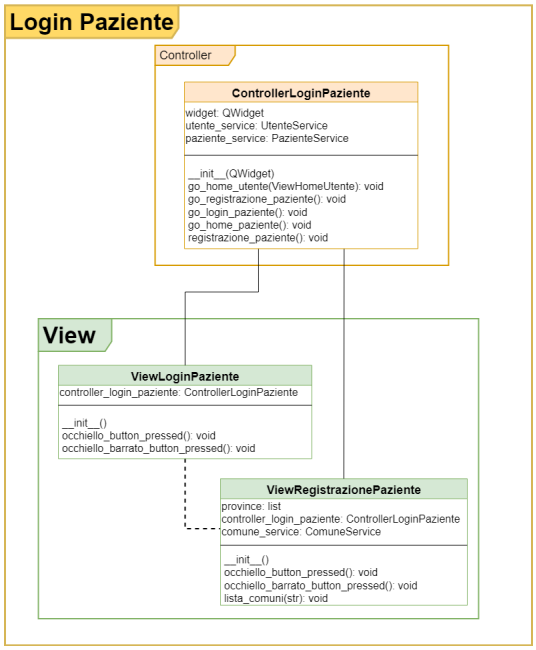

Figura 4.3: Sotto-package Login Paziente

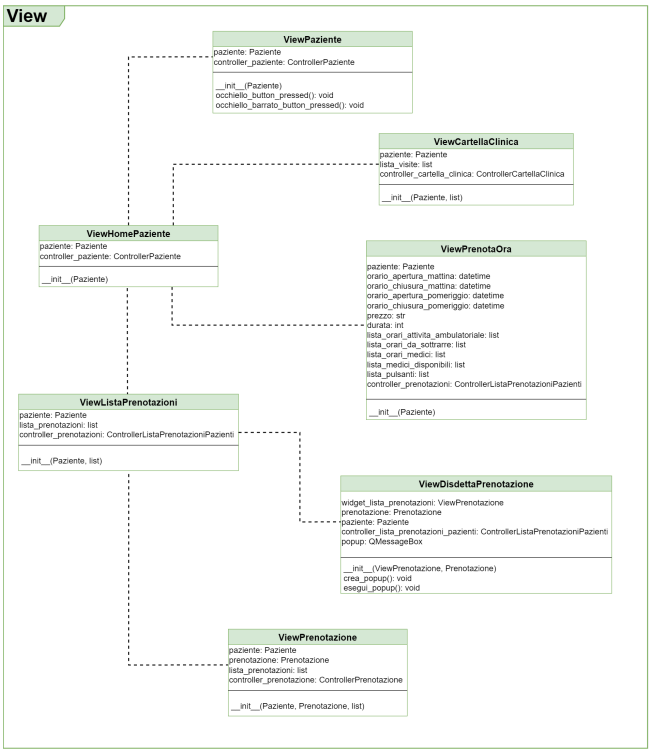

Figura 4.4: Sotto-package View del package Paziente

Sotto-package Controller

In Figura 4.5 è illustrato il sotto-package "Controller", il quale raggruppa le classi di tipo Controller, responsabili della logica di business della parte di sistema relativa al paziente autenticato.

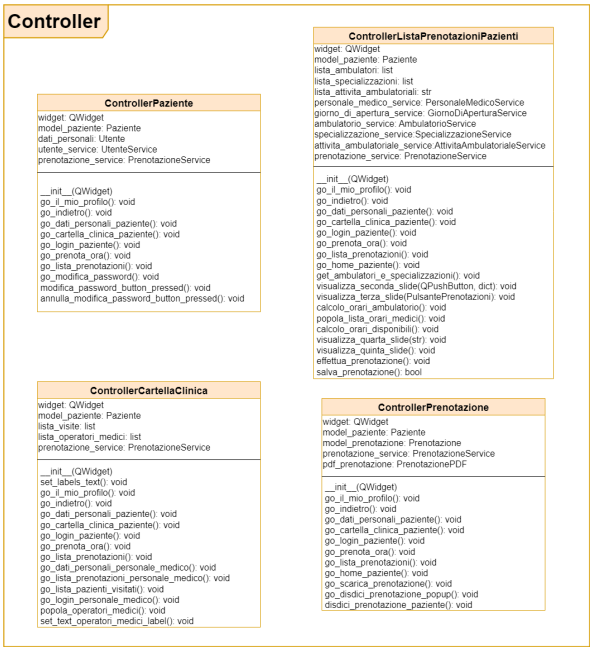

Figura 4.5: Sotto-package Controller del package Paziente

4.2.1.3 Package Personale medico

In Figura 4.6 è illustrato il package "Personale medico", che raggruppa i sotto-package "Login Personale medico", "View" e "Controller". La figura descrive come le singole classi dei diversi package interagiscono tra loro.

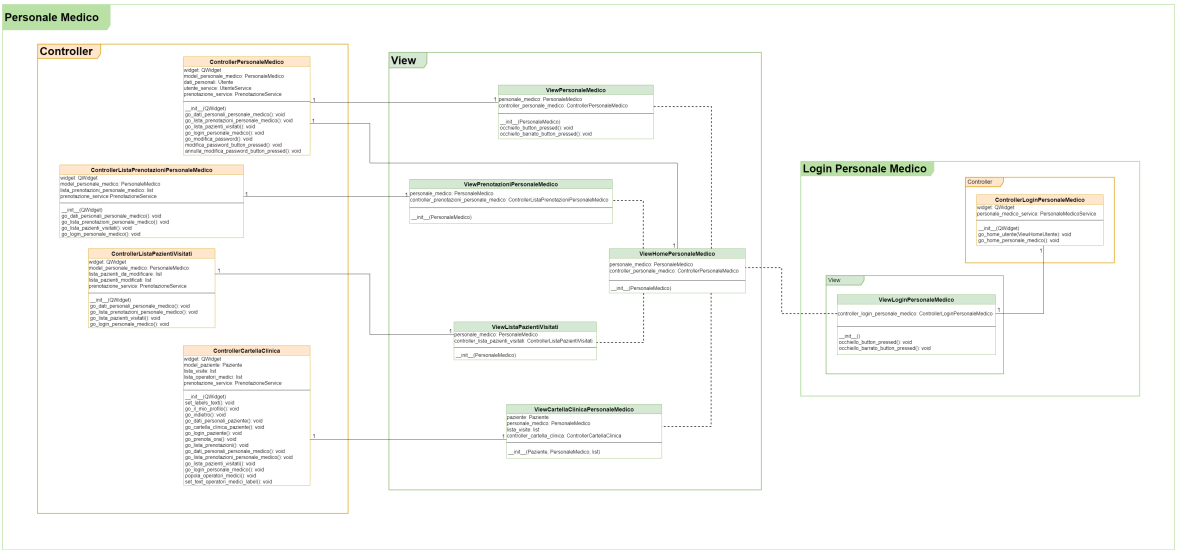

Figura 4.6: Package Personale medico

Descriviamo, ora, i singoli sotto-package.

Sotto-package Login Personale medico

In Figura 4.7 è illustrato il sotto-package "Login Personale medico". Esso raggruppa la View e il Controller che si occupano dell'autenticazione dei membri del personale medico.

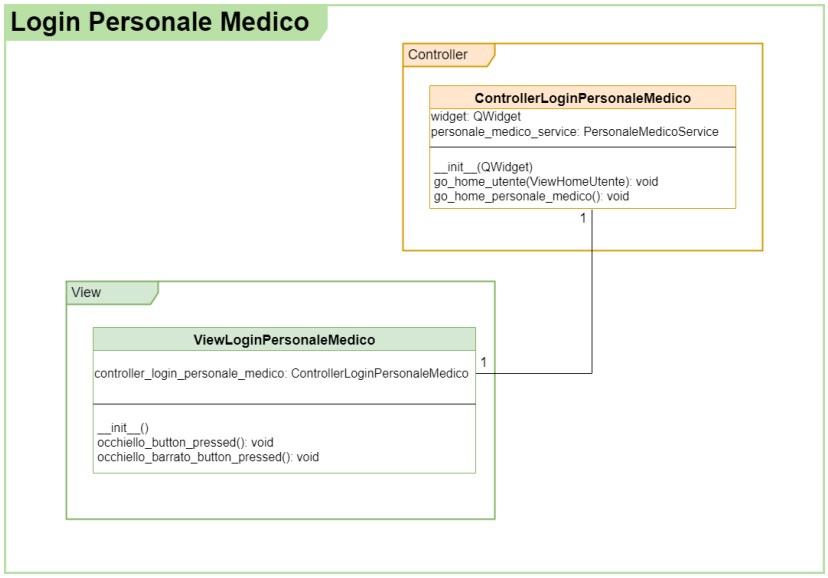

Figura 4.7: Sotto-package Login Personale medico

Sotto-package View

In Figura 4.8 è illustrato il sotto-package "View", il quale raggruppa le classi di tipo View che possono essere mostrate al personale medico autenticato.

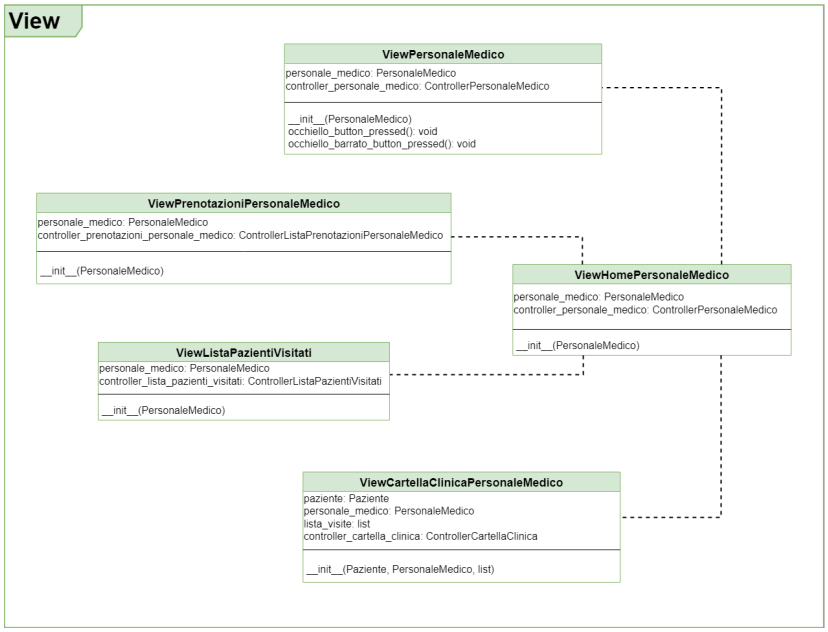

Figura 4.8: Sotto-package View del package Personale medico

Sotto-package Controller

In Figura 4.9 è illustrato il sotto-package "Controller", il quale raggruppa le classi di tipo Controller, responsabili della logica di business della parte di sistema relativa al personale medico autenticato.

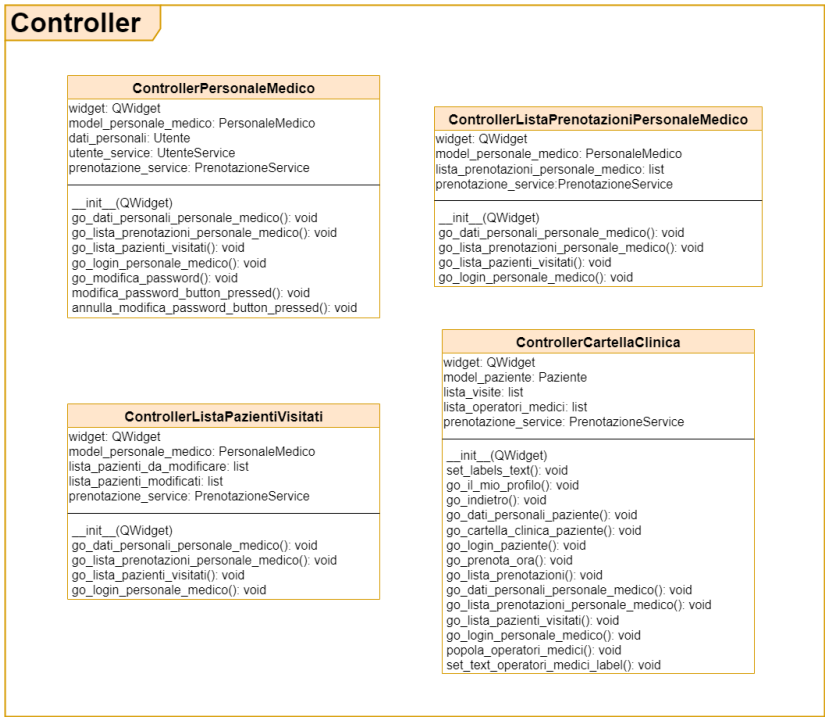

Figura 4.9: Sotto-package Controller del package Personale medico

4.2.1.4 Package Database

In Figura 4.10 è illustrato il package "Database", il quale raggruppa le classi che permettono l'interfacciamento con il database del sistema. Esso contiene:

- la classe "Builder", che crea il database, nel caso in cui esso non esistesse, e successivamente genera le tabelle relative al modello logico descritto nel Capitolo 3;
- la classe "Seeder", che si occupa di riempire di istanze prestabilite le tabelle create dalla classe "Builder";
- il sotto-package "Model", che contiene tutte quelle classi che identificano le tabelle del database;
- il sotto-package "Service", che rappresenta lo strato Service del pattern MVCS. Le classi, contenute all'interno del sotto-package, sono provviste di metodi che consentono il salvataggio e la selezione di istanze dal database ed eventuali aggiornamenti ed eliminazioni.

4.2.1.5 Diagramma completo delle classi

In Figura 4.11 è illustrato il diagramma completo delle classi, che mostra dettagliatamente le associazioni esistenti tra le classi dei vari package, descrivendo anche le cardinalità con cui ogni classe partecipa alle associazioni. Nella parte centrale vi sono i package "Utente" e "Database"; sulla sinistra e sulla destra vi sono, rispettivamente, i package "Paziente" e "Personale medico". Dal diagramma si può notare il flusso degli strati presenti nel pattern MVCS. Le classi di tipo View istanziano i relativi Controller. Essi, oltre a contenere la logica di business dell'applicazione, istanziano le classi di tipo Service che si occupano dell'interfacciamento con il database, "mappando" le istanze delle tabelle con le istanze delle classi di tipo Model.

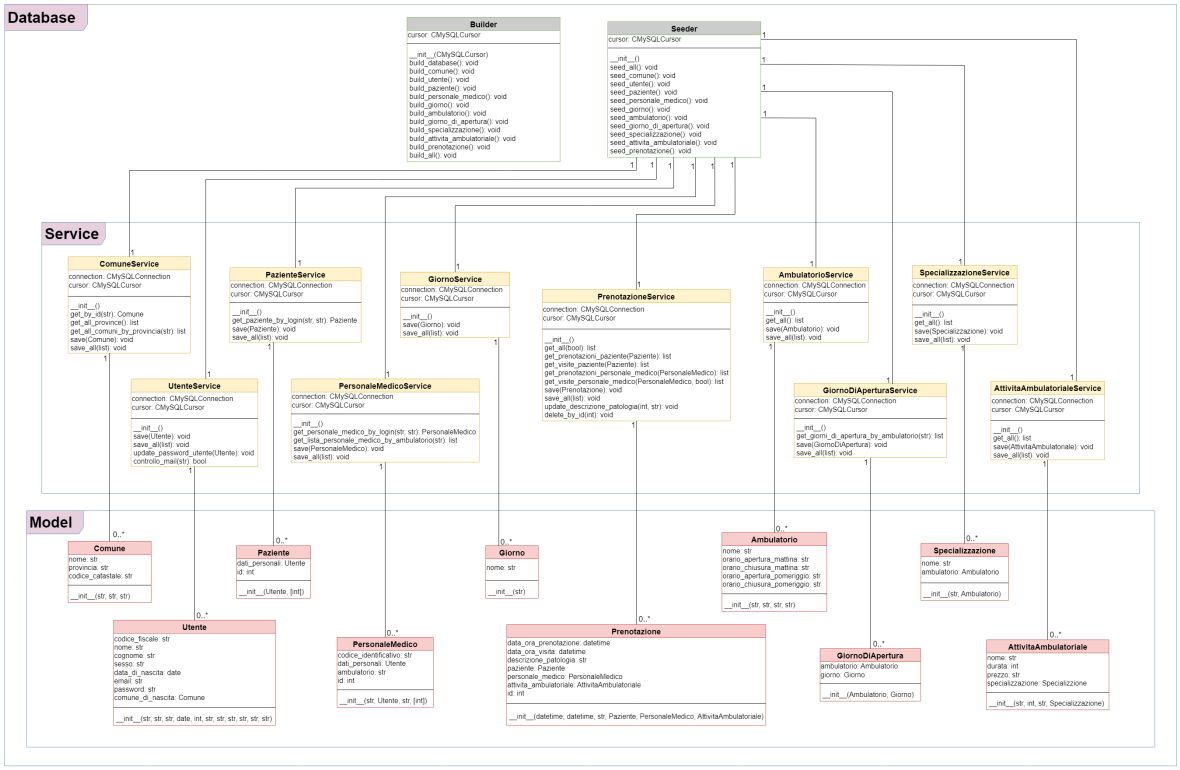

Figura 4.10: Package Database

4.2.1.6 Package Tools

In Figura 4.12 è illustrato il package "Tools" che, per ragioni di leggibilità, non è stato incluso nel diagramma completo delle classi. Esso comprende una serie di classi di utilità generale che facilitano operazioni comuni all'interno del software. Inoltre, il package include:

- classi dedicate alla generazione di oggetti per l'interfaccia grafica;
- classi che gestiscono le animazioni grafiche;
- una classe per la generazione di documenti PDF relativi alle prenotazioni dei pazienti;
- una classe per la generazione del codice fiscale, che è di particolare rilevanza durante la fase di registrazione del paziente.

4.2.2 Diagramma delle attività

Il diagramma delle attività, come quello delle classi, è un tipo di diagramma UML, utilizzato principalmente nell'ambito dell'analisi dei processi per visualizzare graficamente il flusso di attività all'interno di un sistema. Il diagramma delle attività fornisce una rappresentazione visuale dei processi, consentendo agli sviluppatori di comprendere facilmente il flusso delle attività, senza dover analizzare dettagli complicati.

Gli elementi principali dei diagrammi delle attività che verranno utilizzati includono:

- *punti di ingresso e uscita* del flusso delle operazioni;
- *nodi*, ossia azioni o operazioni da compiere;
- *flussi di controllo*, ossia frecce che collegano i vari nodi;

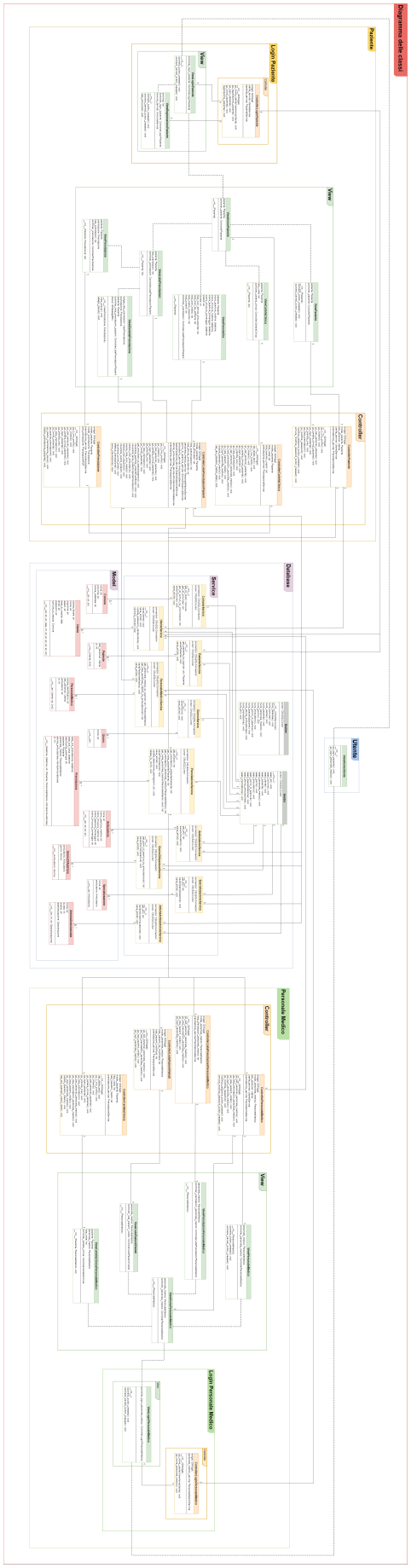

Figura 4.11: Diagramma completo delle classi

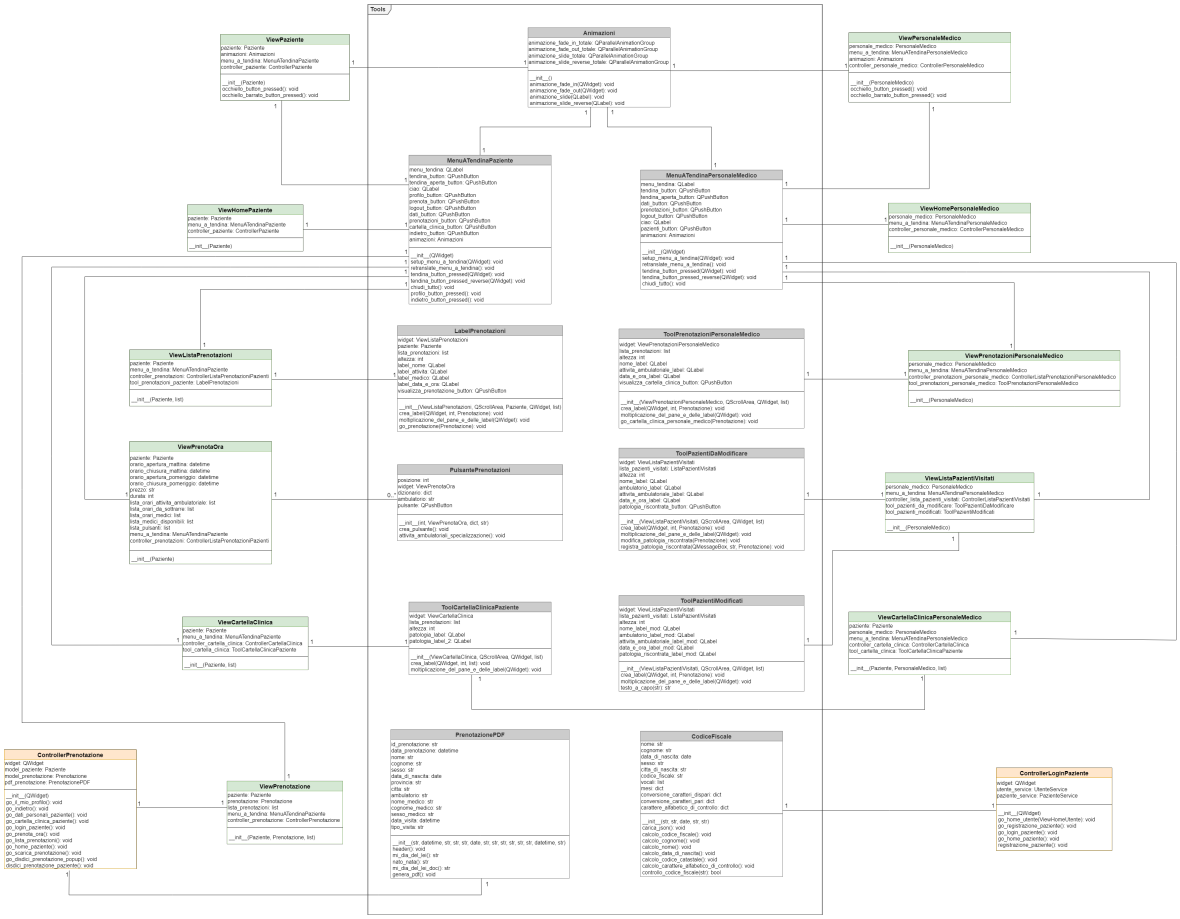

Figura 4.12: Package Tools

- *nodi di decisione*, in cui viene presa una decisione in base a una condizione specifica; a seconda della decisione, il flusso può seguire percorsi alternativi;
- *nodi di biforcazione (fork)*, che consentono l'esecuzione di diverse azioni o decisioni contemporaneamente;
- *nodi di sincronizzazione (join)*, in cui i percorsi, divisi in un nodo di biforcazione, si riuniscono, permettendo di continuare il flusso principale dell'attività;
- *nodi di loop*, in cui le azioni vengono eseguite ripetutamente finché una determinata condizione è soddisfatta.

Esamineremo in dettaglio i diagrammi delle attività dei principali casi d'uso del sistema.

4.2.2.1 Sezione Utente

In Figura 4.13 è illustrato il diagramma delle attività riguardante il caso d'uso "Accedi come paziente".

In Figura 4.14 è illustrato il diagramma delle attività del caso d'uso "Registrazione paziente".

4.2.2.2 Sezione Paziente

In Figura 4.15 è illustrato il diagramma delle attività del caso d'uso "Visualizza informazioni personali del paziente".

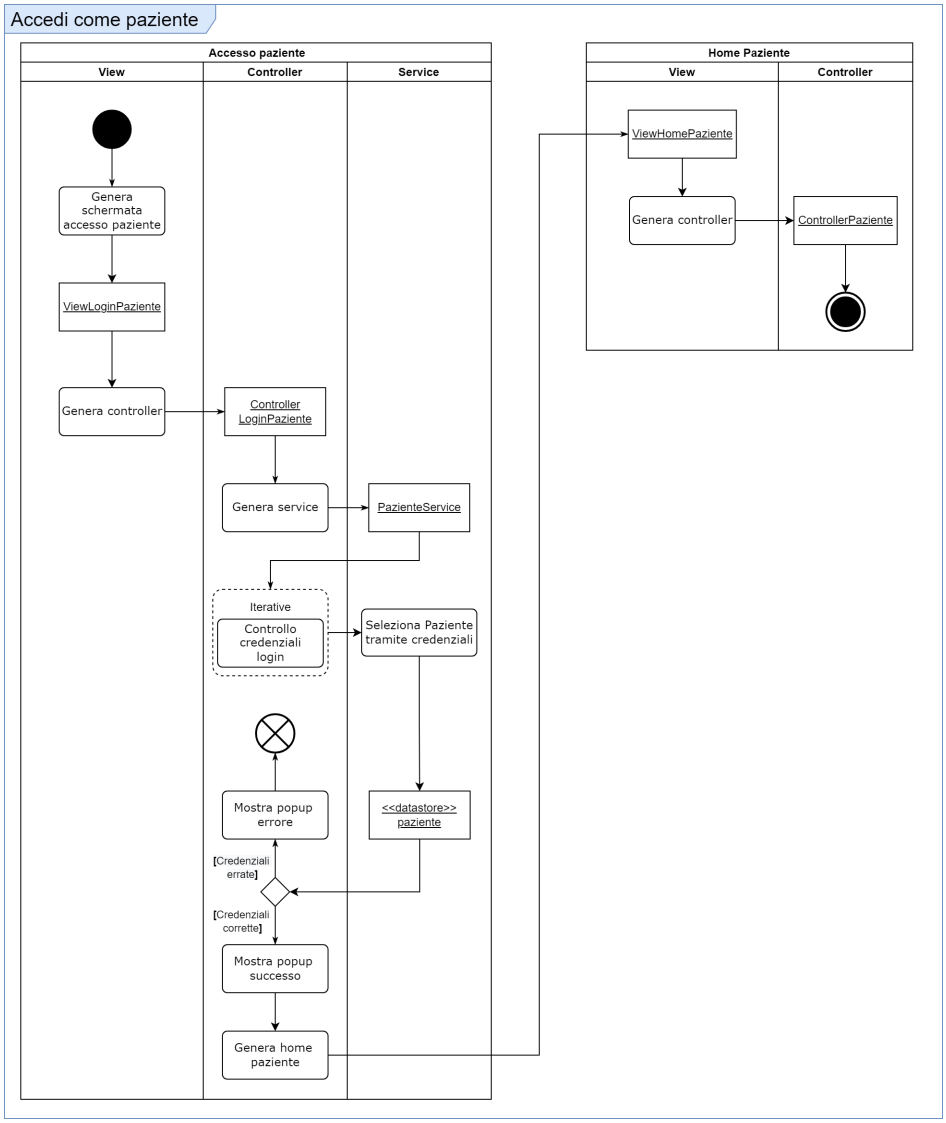

Figura 4.13: Diagramma di attività ″Accedi come paziente″

In Figura 4.16 è illustrato il diagramma delle attività del caso d'uso "Visualizza cartella clinica".

In Figura 4.17 è illustrato il diagramma delle attività del caso d'uso "Visualizza prenotazioni effettuate".

In Figura 4.18 è illustrato il diagramma delle attività del caso d'uso "Prenota nuova visita".

4.2.2.3 Sezione Personale medico

In Figura 4.19 è illustrato il diagramma delle attività del caso d'uso "Visualizza prenotazioni ambulatoriali".

In Figura 4.20 è illustrato il diagramma delle attività del caso d'uso "Visualizza pazienti visitati".

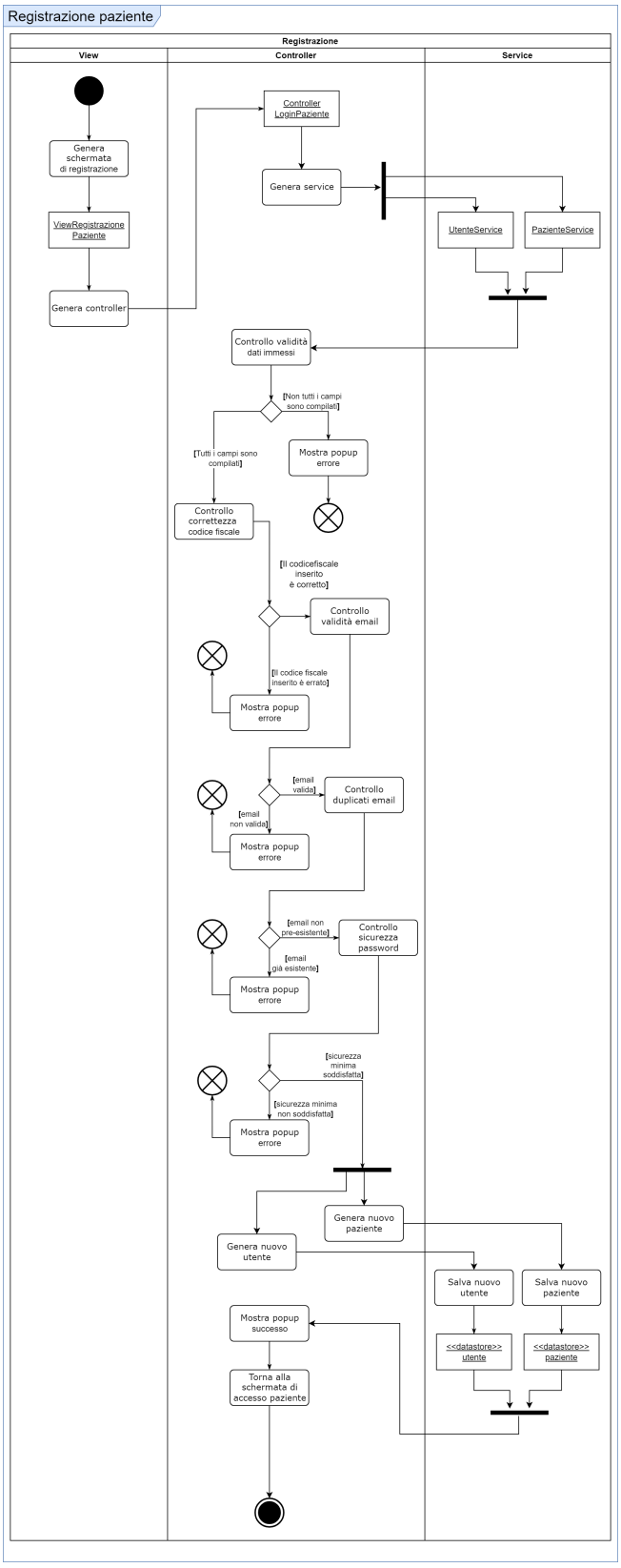

Figura 4.14: Diagramma di attività "Registrazione paziente"

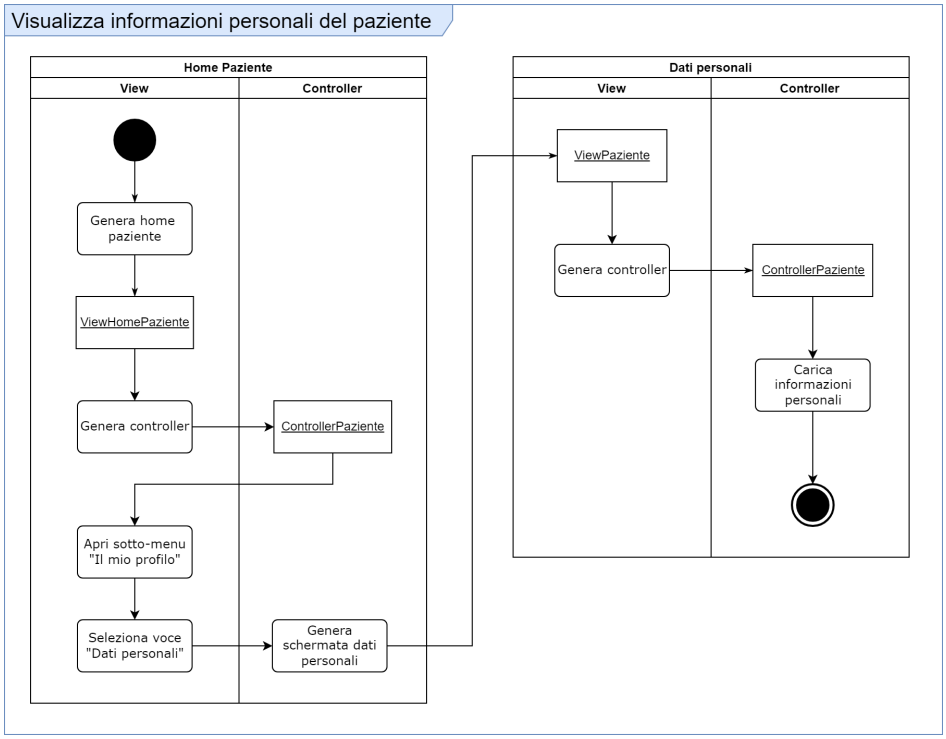

Figura 4.15: Diagramma di attività "Visualizza informazioni personali del paziente"

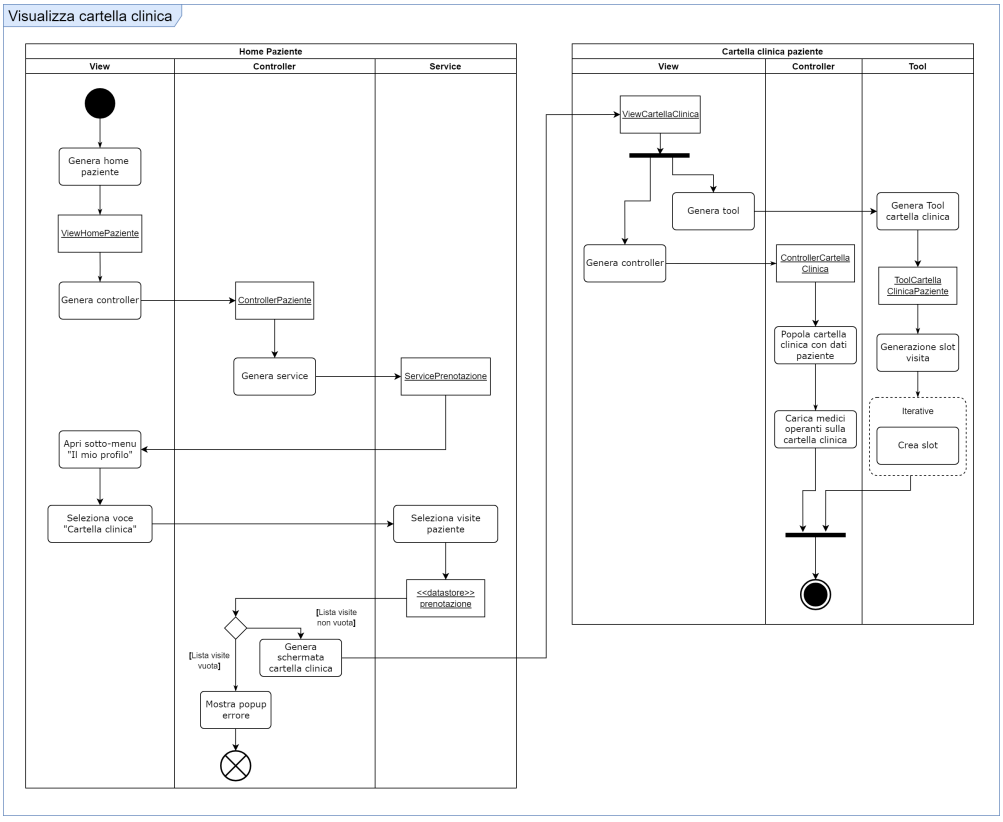

Figura 4.16: Diagramma di attività "Visualizza cartella clinica"

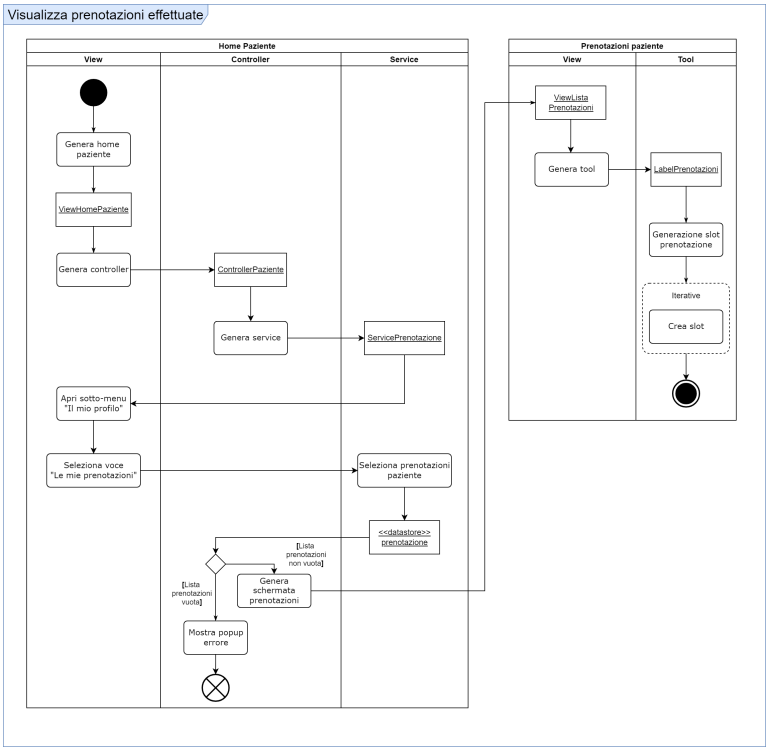

Figura 4.17: Diagramma di attività "Visualizza prenotazioni effettuate"

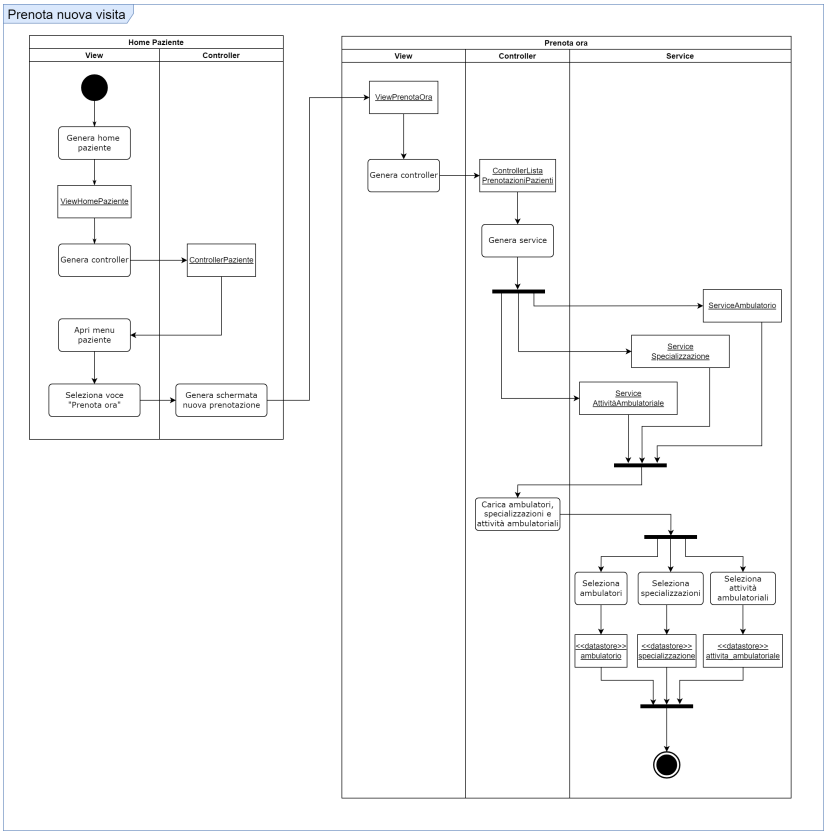

Figura 4.18: Diagramma di attività "Prenota nuova visita"

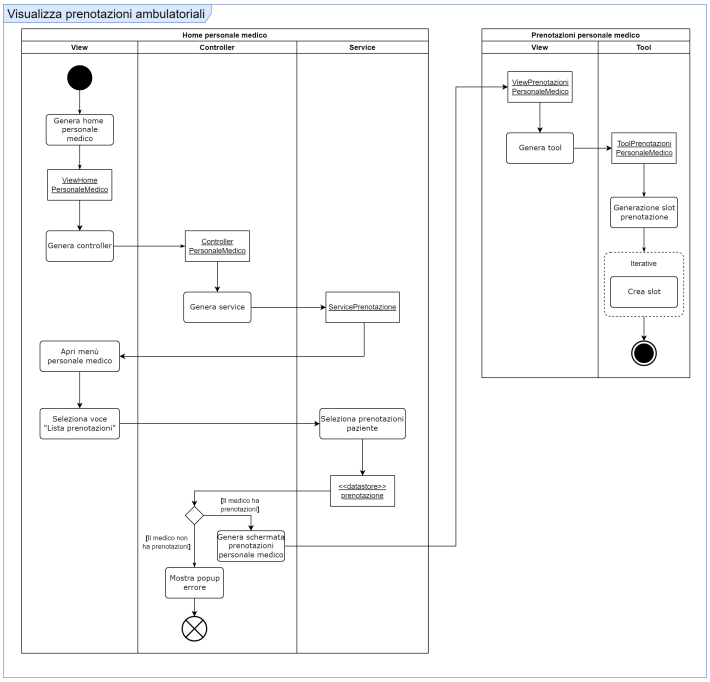

Figura 4.19: Diagramma di attività "Visualizza prenotazioni ambulatoriali"

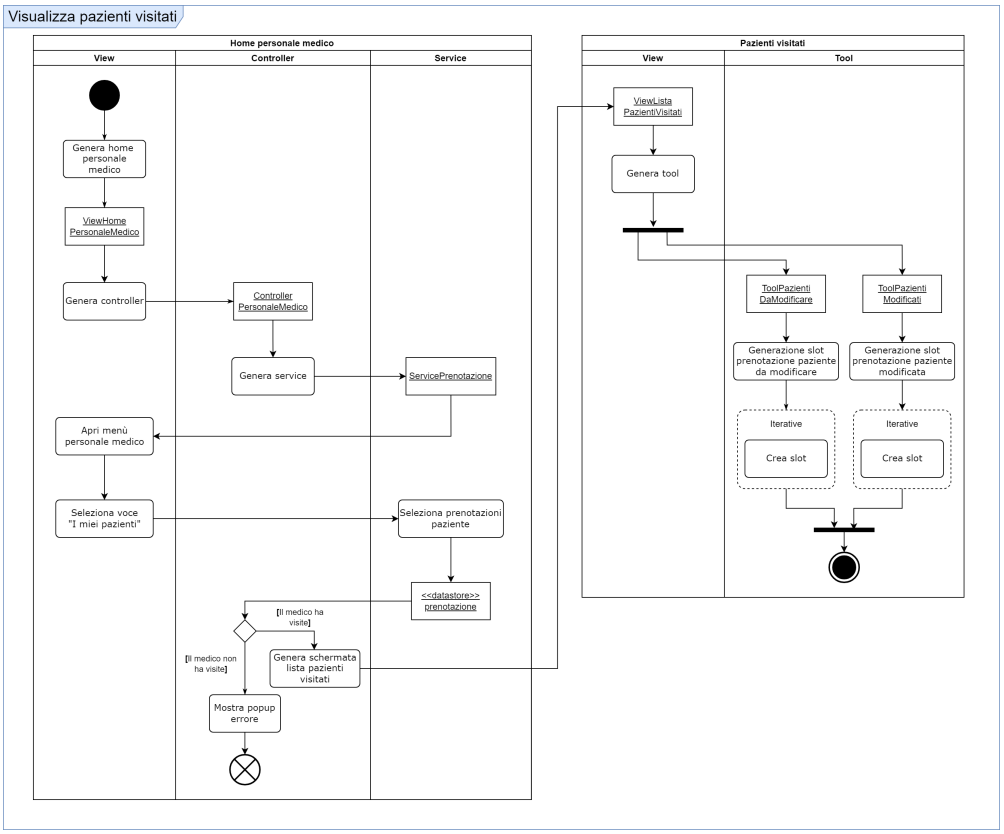

Figura 4.20: Diagramma di attività "Visualizza pazienti visitati"

CAPITOLO 5

Implementazione e manuale utente

In questo capitolo esamineremo l'implementazione del software, fornendo una panoramica delle tecnologie utilizzate e una breve descrizione del modo in cui le diverse componenti del sistema sono state sviluppate e integrate. Successivamente, presenteremo un manuale utente, progettato per orientare nell'uso dell'applicazione, fornendo una visuale delle diverse schermate e delle modalità di esplorazione da parte degli utenti.

5.1 Tecnologie utilizzate

In questa sezione procederemo con la selezione dei componenti necessari per l'implementazione del software e della relativa base di dati, in riferimento alle informazioni descritte nei capitoli precedenti.

Python

Come precedentemente indicato nei requisiti non funzionali, per l'implementazione del software abbiamo optato per l'utilizzo del linguaggio di programmazione "Python". Esso è noto per la sua capacità di rendere il codice sorgente facilmente comprensibile durante le fasi di sviluppo e di manutenzione, consentendo anche a persone diverse da coloro che lo hanno originariamente scritto di apportare modifiche in modo agevole. Inoltre, Python facilita la distribuzione del software su una varietà di sistemi operativi, soddisfacendo, così, il requisito non funzionale "RNF4: Portabilità".

In questo contesto, adotteremo il paradigma di programmazione orientata agli oggetti (OOP), che ci permetterà di definire classi e di sfruttarle in modo efficiente.

Interfaccia grafica

Per la creazione dell'interfaccia grafica dell'applicazione faremo uso del software "Qt Designer". Questo strumento ci consente di definire le interfacce grafiche finali senza la necessità di creare mockup separati. Una volta modellate le schermate dell'applicazione con Qt Designer, potremo tradurle in codice Python utilizzando la libreria "PyQt5".

MySQL

Il sistema di gestione di database relazionali (RDBMS) scelto per la nostra applicazione è "MySQL", basato sul modello relazionale. Esso ci permetterà di implementare il modello logico, descritto nel Capitolo 3, attraverso l'organizzazione dei dati in tabelle relazionali.

5.2 Implementazione software

5.2.1 Implementazione del database

File database_options

Questo file contiene le informazioni necessarie per consentire al sistema di stabilire una connessione con il database. In esso sono specificate le credenziali per un server configurato in un ambiente offline. Tuttavia, è importante notare come il sistema sia compatibile con l'utilizzo di database configurati per operare in modalità online. La sua struttura viene riportata nel Listato 5.1.

```
1  global host, user, password, db_name
2  host="localhost"
3  user="root"
4  password=""
5  db_name = "clinica_privata_casa_alfredo"
```

Listato 5.1: File `database_options`

File connector

Questo file include funzioni dedicate alla gestione della connessione con il database. La prima funzione si occupa della connessione al server mentre la seconda consente di connettersi a un database specifico, il cui nome è dichiarato nel file *database_options*. La sua struttura viene riportata nel Listato 5.2.

```
1  import mysql.connector
2  from database.database_options import *
3
4  def getConnection():
5      connection = mysql.connector.connect(
6          host=host,
7          user=user,
8          password=password
9      )
10     cursor = connection.cursor()
11     return connection, cursor
12
13 def getDatabaseConnection():
14     connection = mysql.connector.connect(
15         host=host,
16         user=user,
17         password=password,
18         database=db_name
19     )
20     cursor = connection.cursor()
21     return connection, cursor
```

Listato 5.2: File `connector`

Classe Builder

La classe *Builder* si occupa della creazione del database e delle relative tabelle. È sufficiente invocare il metodo "build_all", il quale, a sua volta, richiama tutti i metodi necessari per la creazione effettiva del database e delle relative tabelle associate. L'implementazione di questa classe viene mostrata nel Listato 5.3.

```
import database.connector as c

class Builder:

    def __init__(self, cursor):
        self.cursor = cursor

    def build_database(self):
        self.cursor.execute(f"CREATE DATABASE {c.db_name} CHARACTER SET utf8 COLLATE utf8_bin")
        print(f"Database {c.db_name} creato")

    def build_comune(self):
        self.cursor.execute("""
            CREATE TABLE comune(
            nome VARCHAR(255) PRIMARY KEY,
            provincia VARCHAR(255) NOT NULL,
            codice_catastale CHAR(4) NOT NULL CHECK(codice_catastale REGEXP '^[A-Z][0-9]{3}$')
            )"""
        )
        print("Tabella comune creata")

    def build_utente(self):
        self.cursor.execute("""
            CREATE TABLE utente(
            codice_fiscale CHAR(16) PRIMARY KEY CHECK(codice_fiscale REGEXP '^[A-Z0-9]{16}$'),
            nome VARCHAR(255) NOT NULL,
            cognome VARCHAR(255) NOT NULL,
            sesso CHAR(1) NOT NULL CHECK(sesso IN('M','F')),
            data_di_nascita DATE NOT NULL,
            email VARCHAR(255) NOT NULL CHECK(email REGEXP '^[A-Za-z0-9\.\-\_]+@([A-Za-z\-]+\.)+[A-Z-a-z\-]{2,4}$'),
            password VARCHAR(255) NOT NULL,
            comune_di_nascita VARCHAR(255) NOT NULL,
            FOREIGN KEY(comune_di_nascita) REFERENCES comune(nome) ON UPDATE CASCADE ON DELETE NO ACTION
            )"""
        )
        print("Tabella utente creata")

    def build_paziente(self):
        self.cursor.execute("""
            CREATE TABLE paziente(
            id_paziente SMALLINT PRIMARY KEY AUTO_INCREMENT,
            dati_personali CHAR(16) NOT NULL,
            FOREIGN KEY(dati_personali) REFERENCES utente(codice_fiscale) ON UPDATE CASCADE ON DELETE NO ACTION
            )"""
        )
        print("Tabella paziente creata")

    def build_personale_medico(self):
        self.cursor.execute("""
            CREATE TABLE personale_medico(
            id_personale_medico SMALLINT PRIMARY KEY AUTO_INCREMENT,
            codice_identificativo VARCHAR(14) NOT NULL CHECK(codice_identificativo REGEXP '^CPCA[A-Z]{3}pm[0-9]{1,5}$'),
            ambulatorio VARCHAR(30) NOT NULL,
            dati_personali CHAR(16) NOT NULL,
            FOREIGN KEY(dati_personali) REFERENCES utente(codice_fiscale) ON UPDATE CASCADE ON DELETE NO ACTION
            )"""
        )
        print("Tabella personale_medico creata")

    def build_giorno(self):
        self.cursor.execute("""
            CREATE TABLE giorno(
            nome VARCHAR(9) PRIMARY KEY
            )"""
        )
        print("Tabella giorno creata")

    def build_ambulatorio(self):
        self.cursor.execute("""
            CREATE TABLE ambulatorio(
```

```
71              nome VARCHAR(30) PRIMARY KEY,
72              orario_apertura_mattina CHAR(2) NOT NULL,
73              orario_chiusura_mattina CHAR(2) NOT NULL,
74              orario_apertura_pomeriggio CHAR(2) NOT NULL,
75              orario_chiusura_pomeriggio CHAR(2) NOT NULL
76              )"""
77          )
78          print("Tabella ambulatorio creata")
79
80      def build_giorno_di_apertura(self):
81          self.cursor.execute("""
82              CREATE TABLE giorno_di_apertura(
83              ambulatorio VARCHAR(30) NOT NULL,
84              giorno VARCHAR(9) NOT NULL,
85              PRIMARY KEY(ambulatorio, giorno),
86              FOREIGN KEY(ambulatorio) REFERENCES ambulatorio(nome) ON UPDATE CASCADE ON DELETE NO ACTION,
87              FOREIGN KEY(giorno) REFERENCES giorno(nome) ON UPDATE CASCADE ON DELETE NO ACTION
88              )"""
89          )
90          print("Tabella giorno_di_apertura creata")
91
92      def build_specializzazione(self):
93          self.cursor.execute("""
94              CREATE TABLE specializzazione(
95              nome VARCHAR(255) PRIMARY KEY,
96              ambulatorio VARCHAR(30) NOT NULL,
97              FOREIGN KEY(ambulatorio) REFERENCES ambulatorio(nome) ON UPDATE CASCADE ON DELETE NO ACTION
98              )"""
99          )
100         print("Tabella specializzazione creata")
101
102     def build_attivita_ambulatoriale(self):
103         self.cursor.execute("""
104             CREATE TABLE attivita_ambulatoriale(
105             nome VARCHAR(255) PRIMARY KEY,
106             durata SMALLINT NOT NULL,
107             prezzo VARCHAR(4) NOT NULL,
108             specializzazione VARCHAR(255) NOT NULL,
109             FOREIGN KEY(specializzazione) REFERENCES specializzazione(nome) ON UPDATE CASCADE ON DELETE NO ACTION
110             )"""
111         )
112         print("Tabella attivita_ambulatoriale creata")
113
114     def build_prenotazione(self):
115         self.cursor.execute("""
116             CREATE TABLE prenotazione(
117             id_prenotazione INTEGER PRIMARY KEY AUTO_INCREMENT,
118             data_ora_prenotazione DATETIME NOT NULL,
119             data_ora_visita DATETIME NOT NULL,
120             descrizione_patologia VARCHAR(255),
121             paziente SMALLINT NOT NULL,
122             personale_medico SMALLINT NOT NULL,
123             attivita_ambulatoriale VARCHAR(255) NOT NULL,
124             FOREIGN KEY(paziente) REFERENCES paziente(id_paziente) ON UPDATE CASCADE ON DELETE NO ACTION,
125             FOREIGN KEY(personale_medico) REFERENCES personale_medico(id_personale_medico)
126             ON UPDATE CASCADE ON DELETE NO ACTION,
127             FOREIGN KEY(attivita_ambulatoriale) REFERENCES attivita_ambulatoriale(nome)
128             ON UPDATE CASCADE ON DELETE NO ACTION
129             )"""
130         )
131         print("Tabella prenotazione creata")
132
133     def build_all(self):
134         self.build_database()
135         connection, self.cursor = c.getDatabaseConnection()
136
137         self.build_comune()
138         self.build_utente()
139         self.build_paziente()
140         self.build_personale_medico()
141         self.build_giorno()
142         self.build_ambulatorio()
143         self.build_giorno_di_apertura()
144         self.build_specializzazione()
145         self.build_attivita_ambulatoriale()
146         self.build_prenotazione()
```

Listato 5.3: Classe `Builder`

Classe Seeder

La classe *Seeder* ha il compito di popolare tutte le tabelle del database con istanze predefinite. Per effettuare tale operazione è sufficiente invocare il metodo "seed_all". A sua volta, questo metodo ne richiama altri, dedicati all'inserimento di dati specifici nelle singole tabelle. L'implementazione di tale classe viene mostrata nel Listato 5.4.

```
from datetime import date, datetime
import openpyxl
from pathlib import Path

from database.model.comune import Comune
from database.model.utente import Utente
from database.model.paziente import Paziente
from database.model.personale_medico import PersonaleMedico
from database.model.giorno import Giorno
from database.model.ambulatorio import Ambulatorio
from database.model.giorno_di_apertura import GiornoDiApertura
from database.model.specializzazione import Specializzazione
from database.model.attivita_ambulatoriale import AttivitaAmbulatoriale
from database.model.prenotazione import Prenotazione

from database.service.comune_service import ComuneService
from database.service.utente_service import UtenteService
from database.service.paziente_service import PazienteService
from database.service.personale_medico_service import PersonaleMedicoService
from database.service.giorno_service import GiornoService
from database.service.ambulatorio_service import AmbulatorioService
from database.service.giorno_di_apertura_service import GiornoDiAperturaService
from database.service.specializzazione_service import SpecializzazioneService
from database.service.attivita_ambulatoriale_service import AttivitaAmbulatorialeService
from database.service.prenotazione_service import PrenotazioneService

class Seeder:

    def seed_all(self):
        self.seed_comune()
        self.seed_utente()
        self.seed_paziente()
        self.seed_personale_medico()
        self.seed_giorno()
        self.seed_ambulatorio()
        self.seed_giorno_di_apertura()
        self.seed_specializzazione()
        self.seed_attivita_ambulatoriale()
        self.seed_prenotazione()

    def seed_comune(self):
        file = Path('file', 'Province e comuni.xlsx')
        workbook = openpyxl.load_workbook(file)
        sheet = workbook.active

        province_e_comuni = {}
        province = []

        for i, row in enumerate(sheet) :
            if(i==0):
                for cell in row:
                    province.append(cell.value)
                    province_e_comuni[cell.value] = []
            else:
                for i,cell in enumerate(row):
                    if(cell.value != None):
                        province_e_comuni[province[i]].append(cell.value)

        file = Path('file', 'Comuni e codici catastali.xlsx')
        workbook = openpyxl.load_workbook(file)
        sheet = workbook.active

        comuni_e_codici_catastali = {}

        for row in sheet:
            comuni_e_codici_catastali[row[0].value] = row[1].value

        comuni_db = []

        for provincia, comuni in province_e_comuni.items():
```

```
            for comune in comuni:
                comuni_db.append(Comune(comune, provincia, comuni_e_codici_catastali[comune]))

        comune_service = ComuneService()
        comune_service.save_all(comuni_db)

        print("Tabella comune riempita")

    def seed_utente(self):
        utenti = [
            #utenti personale medico
            Utente("RSSMRA70T12G878Q", "Mario", "Rossi", "M", date(1970, 12, 12),
                   "mario.rossi@casa.alfredo.com", "CasaAlfredo1~", "Popoli"),
            Utente("VRDLGU75D27G482A", "Luigi", "Verdi", "M", date(1975, 4, 27),
                   "luigi.verdi@casa.alfredo.com", "CasaAlfredo2~", "Pescara"),
            Utente("BRMNDR78A30D643B", "Andrea", "Brambilla", "M", date(1978, 1, 30),
                   "andrea.brambilla@casa.alfredo.com", "CasaAlfredo3~", "Foggia"),
            Utente("RNCLRA86B50F205V", "Lara", "Ronchi", "F", date(1986, 2, 10),
                   "lara.ronchi@casa.alfredo.com", "CasaAlfredo4~", "Milano"),
            Utente("GLBLCU88C12F205T", "Luca", "Gambussera", "M", date(1988, 3, 12),
                   "luca.galbussera@casa.alfredo.com", "CasaAlfredo5~", "Milano"),
            Utente("FMGFNC88E53L219X", "Francesca", "Fumagalli", "F", date(1988, 5, 13),
                   "francesca.fumagalli@casa.alfredo.com", "CasaAlfredo6~", "Torino"),
            Utente("GRCMRC90E17A271P", "Marco", "Grieco", "M", date(1990, 5, 17),
                   "marco.grieco@casa.alfredo.com", "CasaAlfredo7~", "Ancona"),
            Utente("DSNLRA91H58F839T", "Laura", "De Santis", "F", date(1991, 6, 18),
                   "laura.desantis@casa.alfredo.com", "CasaAlfredo8~", "Napoli"),
            Utente("RCCGNN81P19D612T", "Giovanni", "Ricci", "M", date(1981, 9, 19),
                   "giovanni.ricci@casa.alfredo.com", "CasaAlfredo9~", "Firenze"),
            Utente("MNCLSN87R23C632K", "Alessandro", "Mancini", "M", date(1987, 10, 23),
                   "alessandro.mancini@casa.alfredo.com", "CasaAlfredo10~", "Chieti"),
            Utente("FRRMRT90A42C632D", "Marta", "Ferrari", "F", date(1990, 1, 2),
                   "marta.ferrari@casa.alfredo.com", "CasaAlfredo11~", "Chieti"),
            #utenti paziente
            Utente("RSSMRA90R16C632Y", "Mario", "Rossi", "M", date(1990, 10, 16),
                   "mario.rossi@email.com", "MarioMario.90", "Chieti"),
            Utente("BNCLGU88T21A271P", "Luigi", "Bianchi", "M", date(1988, 12, 21),
                   "luigi.bianchi@email.com", "LuigiLuigi.88", "Ancona"),
            Utente("SPSMRA94S58G878P", "Maria", "Esposito", "F", date(1994, 11, 18),
                   "maria.esposito@email.com", "MariaMaria.94", "Popoli")
        ]

        utente_service = UtenteService()
        utente_service.save_all(utenti)

        print("Tabella utente riempita")

    def seed_paziente(self):
        pazienti = [
            Paziente("RSSMRA90R16C632Y"),
            Paziente("BNCLGU88T21A271P"),
            Paziente("SPSMRA94S58G878P")
        ]

        paziente_service = PazienteService()
        paziente_service.save_all(pazienti)

        print("Tabella paziente riempita")

    def seed_personale_medico(self):
        personale_medico_list = [
            PersonaleMedico("CPCAMGCpm1", "RSSMRA70T12G878Q", "Medicina Generale Cardiologica"),
            PersonaleMedico("CPCAMGCpm2", "VRDLGU75D27G482A", "Medicina Generale Cardiologica"),
            PersonaleMedico("CPCAMGCpm3", "BRMNDR78A30D643B", "Medicina Generale Cardiologica"),
            PersonaleMedico("CPCAENDpm4", "RNCLRA86B50F205V", "Endocrinologia"),
            PersonaleMedico("CPCAUROpm5", "GLBLCU88C12F205T", "Urologia/Andrologia"),
            PersonaleMedico("CPCAUROpm6", "FMGFNC88E53L219X", "Urologia/Andrologia"),
            PersonaleMedico("CPCAOCUpm7", "GRCMRC90E17A271P", "Oculistica"),
            PersonaleMedico("CPCAOCUpm8", "DSNLRA91H58F839T", "Oculistica"),
            PersonaleMedico("CPCAORTpm9","RCCGNN81P19D612T", "Ortopedia"),
            PersonaleMedico("CPCADERpm10", "MNCLSN87R23C632K", "Dermatologia"),
            PersonaleMedico("CPCADERpm11", "FRRMRT90A42C632D", "Dermatologia")
        ]

        personale_medico_service = PersonaleMedicoService()
        personale_medico_service.save_all(personale_medico_list)

        print("Tabella personale_medico riempita")

    def seed_giorno(self):
        giorni = [
```

```
152             Giorno("Lunedì"),
153             Giorno("Martedì"),
154             Giorno("Mercoledì"),
155             Giorno("Giovedì"),
156             Giorno("Venerdì"),
157             Giorno("Sabato"),
158             Giorno("Domenica")
159         ]
160
161         giorno_service = GiornoService()
162         giorno_service.save_all(giorni)
163
164         print("Tabella giorno riempita")
165
166     def seed_ambulatorio(self):
167         ambulatori = [
168             Ambulatorio("Medicina Generale Cardiologica", "07", "13", "15", "19"),
169             Ambulatorio("Endocrinologia", "07", "12", "15", "17"),
170             Ambulatorio("Urologia/Andrologia", "11", "13", "15", "18"),
171             Ambulatorio("Oculistica", "07", "13", "15", "17"),
172             Ambulatorio("Ortopedia", "09", "14", "16", "19"),
173             Ambulatorio("Dermatologia", "07", "13", "15", "19")
174         ]
175
176         ambulatorio_service = AmbulatorioService()
177         ambulatorio_service.save_all(ambulatori)
178
179         print("Tabella ambulatorio riempita")
180
181     def seed_giorno_di_apertura(self):
182         giorni_di_apertura = [
183             GiornoDiApertura("Medicina Generale Cardiologica", "Lunedì"),
184             GiornoDiApertura("Medicina Generale Cardiologica", "Mercoledì"),
185             GiornoDiApertura("Medicina Generale Cardiologica", "Venerdì"),
186             GiornoDiApertura("Endocrinologia", "Martedì"),
187             GiornoDiApertura("Endocrinologia", "Giovedì"),
188             GiornoDiApertura("Urologia/Andrologia", "Mercoledì"),
189             GiornoDiApertura("Urologia/Andrologia", "Giovedì"),
190             GiornoDiApertura("Urologia/Andrologia", "Venerdì"),
191             GiornoDiApertura("Oculistica", "Lunedì"),
192             GiornoDiApertura("Oculistica", "Mercoledì"),
193             GiornoDiApertura("Oculistica", "Sabato"),
194             GiornoDiApertura("Ortopedia", "Martedì"),
195             GiornoDiApertura("Ortopedia", "Venerdì"),
196             GiornoDiApertura("Dermatologia", "Martedì")
197         ]
198
199         giorno_di_apertura_service = GiornoDiAperturaService()
200         giorno_di_apertura_service.save_all(giorni_di_apertura)
201
202         print("Tabella giorno_di_apertura riempita")
203
204     def seed_specializzazione(self):
205         specializzazioni = [
206             Specializzazione("Cardiovascolare", "Medicina Generale Cardiologica"),
207             Specializzazione("Pneumologia", "Medicina Generale Cardiologica"),
208             Specializzazione("Gastroenterologia", "Medicina Generale Cardiologica"),
209             Specializzazione("Neurologia", "Medicina Generale Cardiologica"),
210             Specializzazione("Endocrinologia", "Endocrinologia"),
211             Specializzazione("Urologia/Andrologia", "Urologia/Andrologia"),
212             Specializzazione("Oculistica", "Oculistica"),
213             Specializzazione("Ortopedia", "Ortopedia"),
214             Specializzazione("Dermatologia", "Dermatologia")
215         ]
216
217         specializzazione_service = SpecializzazioneService()
218         specializzazione_service.save_all(specializzazioni)
219
220         print("Tabella specializzazione riempita")
221
222     def seed_attivita_ambulatoriale(self):
223         attivita_ambulatoriale_list = [
224             AttivitaAmbulatoriale("Ecocardiografia", 30, "€110", "Cardiovascolare"),
225             AttivitaAmbulatoriale("ECG basale o da sforzo", 15, "€60", "Cardiovascolare"),
226             AttivitaAmbulatoriale("Holter cardiaco", 15, "€60", "Cardiovascolare"),
227             AttivitaAmbulatoriale("Holter pressorio", 15, "€50", "Cardiovascolare"),
228             AttivitaAmbulatoriale("Visita cardiologica", 30, "€125", "Cardiovascolare"),
229             AttivitaAmbulatoriale("Visita angiologica", 20, "€50", "Cardiovascolare"),
230             AttivitaAmbulatoriale("Visita pneumologica", 20, "€50", "Pneumologia"),
231             AttivitaAmbulatoriale("Visita gastroenterologica", 20, "€150", "Gastroenterologia"),
232             AttivitaAmbulatoriale("Gastroscopia", 15, "€160", "Gastroenterologia"),
```

```
            AttivitaAmbulatoriale("Colonscopia", 30, "€210", "Gastroenterologia"),
            AttivitaAmbulatoriale("Visita neurologica", 30, "€100", "Neurologia"),
            AttivitaAmbulatoriale("Visita endocrinologica", 40, "€130", "Endocrinologia"),
            AttivitaAmbulatoriale("Visita specialistica", 30, "€50", "Endocrinologia"),
            AttivitaAmbulatoriale("Visita urologica", 20, "€100", "Urologia/Andrologia"),
            AttivitaAmbulatoriale("Visita andrologica", 20, "€120", "Urologia/Andrologia"),
            AttivitaAmbulatoriale("Cistoscopia", 10, "€300", "Urologia/Andrologia"),
            AttivitaAmbulatoriale("Ecografia renale", 20, "€40", "Urologia/Andrologia"),
            AttivitaAmbulatoriale("Ecografia vescicale", 20, "€40", "Urologia/Andrologia"),
            AttivitaAmbulatoriale("Ecog. prostatica sovrapubica", 20, "€40", "Urologia/Andrologia"),
            AttivitaAmbulatoriale("Visita oculistica", 30, "€100", "Oculistica"),
            AttivitaAmbulatoriale("Visita ortottica", 30, "€70", "Oculistica"),
            AttivitaAmbulatoriale("Studio del campo visivo", 30, "€150", "Oculistica"),
            AttivitaAmbulatoriale("Perimetria computerizzata", 20, "€130", "Oculistica"),
            AttivitaAmbulatoriale("Pachimetria corneale", 15, "€100", "Oculistica"),
            AttivitaAmbulatoriale("Tomografia corneale", 15, "€100", "Oculistica"),
            AttivitaAmbulatoriale("Visita ortopedica", 20, "€120", "Ortopedia"),
            AttivitaAmbulatoriale("Iniezioni intra-articolari", 10, "€60", "Ortopedia"),
            AttivitaAmbulatoriale("Trattamento rbt. ortopedico", 60, "€45", "Ortopedia"),
            AttivitaAmbulatoriale("Visita dermatologica", 30, "€80", "Dermatologia"),
            AttivitaAmbulatoriale("Biopsie cutanee", 30, "€150", "Dermatologia"),
            AttivitaAmbulatoriale("Mappatura nevi", 30, "€120", "Dermatologia"),
            AttivitaAmbulatoriale("Crioterapia", 10, "€130", "Dermatologia"),
            AttivitaAmbulatoriale("Dermatoscopia", 40, "€80", "Dermatologia")
        ]

        attivita_ambulatoriale_service = AttivitaAmbulatorialeService()
        attivita_ambulatoriale_service.save_all(attivita_ambulatoriale_list)

        print("Tabella attivita_ambulatoriale riempita")

    def seed_prenotazione(self):
        prenotazioni = [
            Prenotazione(datetime(2023, 7, 31, 23, 54, 31), datetime(2023, 8, 28, 9, 15),
                         None, 1, 4, "Visita specialistica"),
            Prenotazione(datetime(2023, 8, 22, 23, 58, 23), datetime(2023, 9, 25, 9, 55),
                         None, 1, 1, "Visita pneumologica")
        ]

        prenotazione_service = PrenotazioneService()
        prenotazione_service.save_all(prenotazioni)

        print("Tabella prenotazione riempita")
```

Listato 5.4: Classe `Seeder`

File repository

Questo file contiene funzioni che vengono eseguite automaticamente ogni volta che l'applicazione viene avviata. La principale responsabilità di queste funzioni è verificare l'esistenza del database; in caso di esito negativo, le funzioni si occupano di creare il database e di riempirlo di istanze, utilizzando i metodi forniti dalle classi *Builder* e *Seeder*. Il codice relativo a tale file viene riportato nel Listato 5.5.

```
import database.connector as c
from database.builder import Builder
from database.seeder import Seeder

def connect():
    connection, cursor = c.getConnection()

    if(check_db_existence(cursor)):
        connection, cursor = c.getDatabaseConnection()
        print(f"Connessione con il database {c.db_name} stabilita")
    else:
        builder = Builder(cursor)
        builder.build_all()
        s = Seeder()
        s.seed_all()

def check_db_existence(cursor):
    cursor.execute("SHOW DATABASES")

```

```
    for database in cursor:
        if(database[0] == c.db_name):
            return True

    return False
```

Listato 5.5: File `repository`

5.2.1.1 Implementazione del Model

Le tabelle del database sono "mappate" singolarmente su classi di tipo ”Model”. In generale, queste classi contengono principalmente un costruttore; tuttavia, può risultare utile definire metodi getter o setter, che vanno oltre la semplice modifica degli attributi interni dell’istanza. Nel Listato 5.6 viene illustrata la classe *Utente*.

```
from PyQt5.QtGui import QIcon, QPixmap
from PyQt5.QtWidgets import QMessageBox

import tools.tool_utente as tool_utente

class Utente:

    def __init__(self, codice_fiscale, nome, cognome, sesso, data_di_nascita, email, password, comune_di_nascita):
        self.codice_fiscale = codice_fiscale
        self.nome = nome
        self.cognome = cognome
        self.sesso = sesso
        self.data_di_nascita = data_di_nascita
        self.email = email
        self.password = password
        self.comune_di_nascita = comune_di_nascita

    def set_nuova_password(self, nuova_password, ripeti_password, vecchia_password):
        popup = QMessageBox()
        icona = QIcon()
        icona.addPixmap(QPixmap("Immagini/logo_casa_alfredo.png"), QIcon.Normal, QIcon.Off)
        popup.setWindowIcon(icona)

        if self.password == vecchia_password:
            if self.password == nuova_password:
                popup.setWindowTitle('Attenzione')
                popup.setIcon(QMessageBox.Warning)
                popup.setText('La nuova password deve essere diversa dalla vecchia password!')
                popup.exec_()
            else:
                if nuova_password == ripeti_password:
                    if tool_utente.controllo_sicurezza_password(nuova_password):
                        self.password = nuova_password
                        popup.setWindowTitle('Attenzione')
                        popup.setIcon(QMessageBox.Information)
                        popup.setText('Password aggiornata con successo!')
                        popup.exec_()
                        return True
                    else:
                        popup.setWindowTitle('Attenzione')
                        popup.setIcon(QMessageBox.Warning)
                        popup.setText('La password non è sicura! Seguire le indicazioni.')
                        popup.exec_()
                else:
                    popup.setWindowTitle('Attenzione')
                    popup.setIcon(QMessageBox.Warning)
                    popup.setText('La nuova password inserita e la sua conferma non coincidono!')
                    popup.exec_()
        else:
            popup.setWindowTitle('Attenzione')
            popup.setIcon(QMessageBox.Warning)
            popup.setText('La vecchia password inserita non è corretta')
            popup.exec_()

        return False
```

Listato 5.6: Classe `Utente`

5.2.1.2 Implementazione del Service

Lo strato Service del pattern MVCS è rappresentato dalle classi presenti all'interno del sotto-package "service" del package "database". Queste classi contengono metodi per eseguire operazioni CRUD (Create, Read, Update e Delete). In pratica, le classi di tipo "Service" formano un ORM (Object-Relational Mapping), semplificando notevolmente il lavoro del programmatore. Una volta definiti i metodi CRUD, il programmatore non deve più gestire direttamente le query SQL. Esiste una classe di tipo "Service" per ogni classe di tipo "Model".

Nel caso di query di tipo SELECT, il service traduce le istanze del database in oggetti di tipo "Model". Gli attributi di questi oggetti, che nel modello relazionale rappresentano delle chiavi esterne, vengono a loro volta "mappati" da oggetti. Questo approccio consente al programmatore di accedere agli attributi di un'istanza utilizzando semplicemente la "dot notation" (notazione col punto).

Per illustrare questo concetto, nel Listato 5.7 viene illustrata la classe *UtenteService*, mentre nel Listato 5.8 viene descritto in particolare il metodo "get_paziente_by_login", della classe *PazienteService*. Quest'ultimo mostra come avviene la traduzione di un'istanza della tabella "paziente" in un oggetto della classe *Paziente*.

```python
import database.connector as c

class UtenteService:

    def __init__(self):
        self.connection, self.cursor = c.getDatabaseConnection()

    def save(self, utente):
       query = """
            INSERT INTO `utente`(`codice_fiscale`, `nome`, `cognome`, `sesso`, `data_di_nascita`,
            `email`, `password`, `comune_di_nascita`) VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
            """

        values = (utente.codice_fiscale, utente.nome, utente.cognome, utente.sesso,
                  utente.data_di_nascita, utente.email, utente.password, utente.comune_di_nascita)

        self.cursor.execute(query, values)
        self.connection.commit()

    def save_all(self, utenti):
        query = """
            INSERT INTO `utente`(`codice_fiscale`, `nome`, `cognome`, `sesso`, `data_di_nascita`,
            `email`, `password`, `comune_di_nascita`) VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
            """
        values = []

        for utente in utenti:
            values.append((utente.codice_fiscale, utente.nome, utente.cognome, utente.sesso,
                           utente.data_di_nascita, utente.email, utente.password,
                           utente.comune_di_nascita))

        self.cursor.executemany(query, values)
        self.connection.commit()

    def update_password_utente(self, utente):
        query = f"UPDATE `utente` SET `password`='{utente.password}' WHERE codice_fiscale='{utente.codice_fiscale}'"
        self.cursor.execute(query)
        self.connection.commit()

    def controllo_mail(self, email):
        query = f"SELECT * FROM utente WHERE email='{email}'"
        self.cursor.execute(query)
        riga = self.cursor.fetchone()

        if riga == None:
            return False
        else:
            return True
```

Listato 5.7: Classe `UtenteService`

```
def get_paziente_by_login(self, email, password):
        query = f"""
            SELECT * FROM (paziente LEFT JOIN utente ON paziente.dati_personali = utente.codice_fiscale)
            LEFT JOIN comune ON utente.comune_di_nascita = comune.nome
            WHERE utente.email = '{email}' AND utente.password = '{password}'
            """
        self.cursor.execute(query)
        riga = self.cursor.fetchone()
        if(riga == None):
            return riga
        else:
            p = Paziente(Utente(riga[2], riga[3], riga[4], riga[5], riga[6], riga[7], riga[8],
                        Comune(riga[10], riga[11], riga[12])), riga[0])
        return p
```

Listato 5.8: Metodo `get_paziente_by_login` della classe `PazienteService`

5.2.2 Implementazione del Controller

Le classi di tipo "Controller" contengono la logica di business dell'applicazione e gestiscono le informazioni dinamiche del sistema, riflettendole sull'interfaccia grafica.

Inoltre, queste classi sono responsabili della gestione degli eventi che gli utenti possono generare, come, ad esempio, premere un pulsante o interagire con l'interfaccia. Nella maggior parte dei casi, i metodi all'interno di queste classi sono progettati per coordinare il cambiamento delle schermate dell'applicazione, guidando, così, l'interazione utente.

Nel Listato 5.9 viene presentata la classe *ControllerPaziente* per illustrare come questa logica di gestione venga implementata in pratica.

```
import datetime
from PyQt5.QtGui import QIcon, QPixmap
from PyQt5.QtWidgets import QMessageBox
from database.service.prenotazione_service import PrenotazioneService
from database.service.utente_service import UtenteService

from paziente.login_paziente.view import view_login_paziente
from paziente.view.view_cartella_clinica_paziente import ViewCartellaClinica
from paziente.view.view_dati_personali_paziente import ViewPaziente
from paziente.view.view_prenota_ora import ViewPrenotaOra
from paziente.view.view_prenotazioni_paziente import ViewListaPrenotazioni

import tools.tool_utente as tool_utente

class ControllerPaziente:

    def __init__(self, widget):
        self.widget = widget
        self.model_paziente = widget.paziente
        self.dati_personali = self.model_paziente.dati_personali

        self.utente_service = UtenteService()
        self.prenotazione_service = PrenotazioneService()

        if isinstance(widget, ViewPaziente):
            widget.occhiello_button.clicked.connect(widget.occhiello_button_pressed)
            widget.occhiello_barrato_button.clicked.connect(widget.occhiello_barrato_button_pressed)
            widget.modifica_password.clicked.connect(self.modifica_password_button_pressed)
            widget.annulla_modifica_password.clicked.connect(self.annulla_modifica_password_button_pressed)

            date_time_to_str = self.dati_personali.data_di_nascita.strftime("%d/%m/%Y")
            eta = str(tool_utente.calcolo_eta(self.dati_personali.data_di_nascita))

            widget.nome_label.setText(self.dati_personali.nome)
            widget.cognome_label.setText(self.dati_personali.cognome)
            widget.data_di_nascita_label.setText(date_time_to_str)
            widget.eta_label.setText(eta)
            widget.sesso_label.setText(self.dati_personali.sesso)
            widget.provincia_label.setText(self.dati_personali.comune_di_nascita.provincia)
            widget.comune_di_nascita_label.setText(self.dati_personali.comune_di_nascita.nome)
```

```
            widget.codice_fiscale_label.setText(self.dati_personali.codice_fiscale)
            widget.email_label.setText(self.dati_personali.email)
            widget.password_line_edit.setText(self.dati_personali.password)
            widget.conferma_button.clicked.connect(self.go_modifica_password)
            widget.conferma_password_line_edit.returnPressed.connect(self.go_modifica_password)

        widget.menu_a_tendina.profilo_button.clicked.connect(self.go_il_mio_profilo)
        widget.menu_a_tendina.indietro_button.clicked.connect(self.go_indietro)
        widget.menu_a_tendina.dati_button.clicked.connect(self.go_dati_personali_paziente)
        widget.menu_a_tendina.cartella_clinica_button.clicked.connect(self.go_cartella_clinica_paziente)
        widget.menu_a_tendina.logout_button.clicked.connect(self.go_login_paziente)
        widget.menu_a_tendina.prenotazioni_button.clicked.connect(self.go_lista_prenotazioni)
        widget.menu_a_tendina.prenota_button.clicked.connect(self.go_prenota_ora)

    def go_il_mio_profilo(self):
        self.widget.menu_a_tendina.profilo_button_pressed()

    def go_indietro(self):
        self.widget.menu_a_tendina.indietro_button_pressed()

    def go_dati_personali_paziente(self):
        dati_personali = ViewPaziente(self.model_paziente)
        dati_personali.show()
        self.widget.hide()

    def go_cartella_clinica_paziente(self):
        lista_visite = self.prenotazione_service.get_visite_paziente(self.model_paziente)
        cartella_clinica_paziente = ViewCartellaClinica(self.model_paziente, lista_visite)

        if not lista_visite:
            popup = QMessageBox()
            icona = QIcon()
            icona.addPixmap(QPixmap("Immagini/logo_casa_alfredo.png"), QIcon.Normal, QIcon.Off)
            popup.setWindowIcon(icona)
            popup.setWindowTitle('Attenzione')
            popup.setIcon(QMessageBox.Warning)
            popup.setText('Non hai mai effettuato visite nei nostri ambulatori!')
            popup.exec_()
        else:
            cartella_clinica_paziente.show()
            cartella_clinica_paziente.controller_cartella_clinica.set_labels_text()
            self.widget.hide()

    def go_login_paziente(self):
        login_paziente = view_login_paziente.ViewLoginPaziente()
        login_paziente.show()
        self.widget.hide()

    def go_prenota_ora(self):
        prenota_ora = ViewPrenotaOra(self.model_paziente)
        prenota_ora.show()
        self.widget.hide()

    def go_lista_prenotazioni(self):
        lista_prenotazioni = self.prenotazione_service.get_prenotazioni_paziente(self.model_paziente)
        vista_prenotazioni_paziente = ViewListaPrenotazioni(self.model_paziente, lista_prenotazioni)

        if not lista_prenotazioni:
            popup = QMessageBox()
            icona = QIcon()
            icona.addPixmap(QPixmap("Immagini/logo_casa_alfredo.png"), QIcon.Normal, QIcon.Off)
            popup.setWindowIcon(icona)
            popup.setWindowTitle('Attenzione')
            popup.setIcon(QMessageBox.Warning)
            popup.setText('Non hai effettuato nessuna prenotazione!')
            popup.exec_()
        else:
            vista_prenotazioni_paziente.show()
            self.widget.hide()

    def go_modifica_password(self):
        if (not self.widget.vecchia_password_line_edit.text()
                or not self.widget.nuova_password_line_edit.text()
                or not self.widget.conferma_password_line_edit.text()):
            popup = QMessageBox()
            icona = QIcon()
            icona.addPixmap(QPixmap("Immagini/logo_casa_alfredo.png"), QIcon.Normal, QIcon.Off)
            popup.setWindowIcon(icona)
            popup.setWindowTitle('Attenzione')
            popup.setIcon(QMessageBox.Warning)
            popup.setText('Riempire accuratamente tutti i campi!')
```

```
122             popup.exec_()
123         else:
124             if self.dati_personali.set_nuova_password(self.widget.nuova_password_line_edit.text(),
125                                                       self.widget.conferma_password_line_edit.text(),
126                                                       self.widget.vecchia_password_line_edit.text()):
127 
128                 self.utente_service.update_password_utente(self.model_paziente.dati_personali)
129                 self.go_login_paziente()
130 
131     def modifica_password_button_pressed(self):
132         self.widget.modifica_password.setVisible(False)
133         self.widget.annulla_modifica_password.setVisible(True)
134         self.widget.modifica_password_label.setVisible(True)
135         self.widget.vecchia_password.setVisible(True)
136         self.widget.nuova_password.setVisible(True)
137         self.widget.confema_password.setVisible(True)
138         self.widget.vecchia_password_line_edit.setVisible(True)
139         self.widget.nuova_password_line_edit.setVisible(True)
140         self.widget.conferma_password_line_edit.setVisible(True)
141         self.widget.scritta.setVisible(True)
142         self.widget.conferma_button.setVisible(True)
143 
144         self.widget.animazioni.animazione_fade_in_totale.clear()
145 
146         self.widget.animazioni.animazione_fade_in(self.widget.modifica_password_label)
147         self.widget.animazioni.animazione_fade_in(self.widget.vecchia_password)
148         self.widget.animazioni.animazione_fade_in(self.widget.nuova_password)
149         self.widget.animazioni.animazione_fade_in(self.widget.confema_password)
150         self.widget.animazioni.animazione_fade_in(self.widget.vecchia_password_line_edit)
151         self.widget.animazioni.animazione_fade_in(self.widget.nuova_password_line_edit)
152         self.widget.animazioni.animazione_fade_in(self.widget.conferma_password_line_edit)
153         self.widget.animazioni.animazione_fade_in(self.widget.scritta)
154         self.widget.animazioni.animazione_fade_in(self.widget.conferma_button)
155 
156         self.widget.animazioni.animazione_fade_in_totale.start()
157 
158     def annulla_modifica_password_button_pressed(self):
159         self.widget.modifica_password.setVisible(True)
160         self.widget.annulla_modifica_password.setVisible(False)
161 
162         self.widget.animazioni.animazione_fade_out_totale.clear()
163 
164         self.widget.animazioni.animazione_fade_out(self.widget.modifica_password_label)
165         self.widget.animazioni.animazione_fade_out(self.widget.vecchia_password)
166         self.widget.animazioni.animazione_fade_out(self.widget.nuova_password)
167         self.widget.animazioni.animazione_fade_out(self.widget.confema_password)
168         self.widget.animazioni.animazione_fade_out(self.widget.vecchia_password_line_edit)
169         self.widget.animazioni.animazione_fade_out(self.widget.nuova_password_line_edit)
170         self.widget.animazioni.animazione_fade_out(self.widget.conferma_password_line_edit)
171         self.widget.animazioni.animazione_fade_out(self.widget.scritta)
172         self.widget.animazioni.animazione_fade_out(self.widget.conferma_button)
173 
174         self.widget.animazioni.animazione_fade_out_totale.start()
175 
176         self.widget.vecchia_password_line_edit.clear()
177         self.widget.nuova_password_line_edit.clear()
178         self.widget.conferma_password_line_edit.clear()
```

Listato 5.9: Classe `ControllerPaziente`

5.2.3 Implementazione della View

Le classi di tipo "View" rappresentano le diverse schermate dell'applicazione che gli utenti possono navigare. Ogni schermata è caratterizzata da elementi grafici specifici i quali vengono creati all'interno del costruttore della classe e, successivamente, configurati nel metodo `setup_ui`. Inoltre, a ciascuna vista è associata un'istanza di una classe di tipo "Controller", la quale gestisce le informazioni non predefinite che devono essere visualizzate e le azioni che l'utente può compiere all'interno della vista stessa. Il testo statico associato a ciascuna vista viene solitamente impostato nel metodo `retranslate_ui`. Nel Listato 5.10 viene illustrata la classe *ViewHomePaziente*.

```
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QWidget

from paziente.controller.controller_paziente import ControllerPaziente

from tools.menu_a_tendina_paziente import MenuATendinaPaziente
from tools import ridimensiona_widget

class ViewHomePaziente(QWidget):

    def __init__(self, paziente):
        super().__init__()

        self.paziente = paziente

        self.logo = QtWidgets.QLabel(self)
        self.titolo = QtWidgets.QLabel(self)
        self.scritta = QtWidgets.QLabel(self)
        self.setup_ui(self)

        self.menu_a_tendina = MenuATendinaPaziente(self)

        self.controller_paziente = ControllerPaziente(self)

        ridimensiona_widget.ridimensiona_view(self)

    def setup_ui(self, home_paziente):
        home_paziente.setObjectName("home_paziente")
        home_paziente.move(0, 0)
        home_paziente.resize(1920, 1080)
        home_paziente.setMinimumSize(QtCore.QSize(1920, 1080))
        home_paziente.setMaximumSize(QtCore.QSize(1920, 1080))
        palette = QtGui.QPalette()
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 255))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Button, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 255))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Base, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 255))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.Window, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 255))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Button, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 255))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Base, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 255))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.Window, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 255))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Button, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 255))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Base, brush)
        brush = QtGui.QBrush(QtGui.QColor(255, 255, 255))
        brush.setStyle(QtCore.Qt.SolidPattern)
        palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.Window, brush)
        home_paziente.setPalette(palette)
        icon = QtGui.QIcon()
        icon.addPixmap(QtGui.QPixmap("Immagini/logo_casa_alfredo.png"), QtGui.QIcon.Normal, QtGui.QIcon.Off)
        home_paziente.setWindowIcon(icon)
        home_paziente.setStyleSheet("")
        self.logo.setGeometry(QtCore.QRect(685, 62, 550, 620))
        self.logo.setText("")
        self.logo.setPixmap(QtGui.QPixmap("Immagini/logo_casa_alfredo.png"))
        self.logo.setScaledContents(True)
        self.logo.setObjectName("logo")
        self.titolo.setGeometry(QtCore.QRect(410, 620, 1100, 140))
        self.titolo.setText("")
        self.titolo.setPixmap(QtGui.QPixmap("Immagini/titolo_casa_alfredo.png"))
        self.titolo.setScaledContents(True)
        self.titolo.setObjectName("titolo")
        self.scritta.setGeometry(QtCore.QRect(695, 820, 530, 130))
        font = QtGui.QFont()
        font.setFamily("Verdana")
        font.setPixelSize(24)
        self.scritta.setFont(font)
```

```
81          self.scritta.setAlignment(QtCore.Qt.AlignHCenter | QtCore.Qt.AlignTop)
82          self.scritta.setObjectName("scritta")
83
84          self.retranslate_ui(home_paziente)
85          QtCore.QMetaObject.connectSlotsByName(home_paziente)
86
87      def retranslate_ui(self, home_paziente):
88          _translate = QtCore.QCoreApplication.translate
89          home_paziente.setWindowTitle(_translate("home_paziente", "Clinica Casa Alfredo"))
90          self.scritta.setText(_translate("home_paziente", "Via Don Luigi Sturzo, 27, Roccamorice(PE)\n"
91                                                           "Telefono: 085 875 2684\n"
92                                                           "Fax: 085 132 0297\n"
93                                                           "E-mail: info@casalfredo.it"))
```

Listato 5.10: Classe `ViewHomePaziente`

5.3 Manuale utente

5.3.1 Sezione Utente

In Figura 5.1 è illustrata la home page dell'utente generico da cui è possibile scegliere se effettuare l'autenticazione come paziente o come membro del personale medico.

Figura 5.1: Vista relativa alla home page dell'utente

Se l'utente seleziona il pulsante "Paziente", viene visualizzata la schermata di autenticazione dedicata ai pazienti, come illustrato nella Figura 5.2. Al contrario, se l'utente seleziona il pulsante "Personale Medico", esso viene indirizzato alla schermata di autenticazione riservata al personale medico, come mostrato nella Figura 5.3.

Nella schermata di autenticazione del paziente, oltre alla possibilità di accedere al sistema, un utente non registrato può effettuare la registrazione premendo il pulsante "Registrati". Verrà, così, visualizzata la schermata relativa alla registrazione del paziente, come illustrato nella Figura 5.4. Questa opzione non è disponibile nella schermata di autenticazione del personale medico.

Premendo il tasto "Registrati ora" l'utente viene registrato nel sistema come paziente e potrà, successivamente, accedere con le credenziali scelte in fase di registrazione.

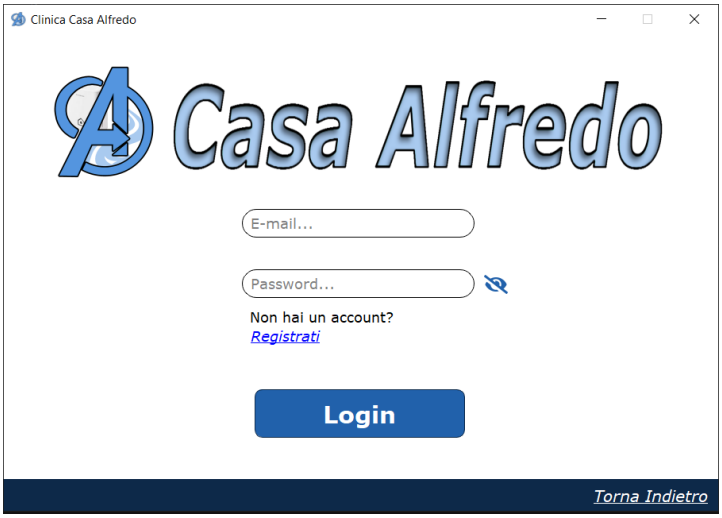

Figura 5.2: Vista relativa al login del paziente

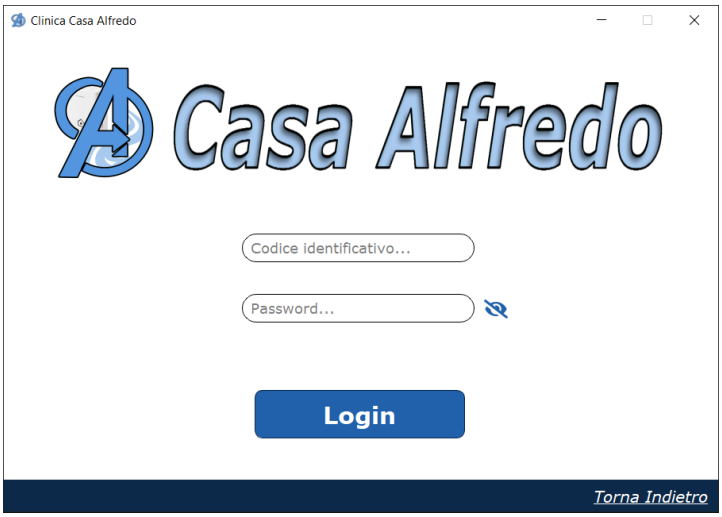

Figura 5.3: Vista relativa al login del personale medico

Figura 5.4: Vista relativa alla registrazione del paziente

5.3.2 Sezione Paziente

Una volta effettuato l'accesso, il paziente visualizza la schermata in Figura 5.5. Da qui egli può interagire con il menù a tendina che mostra le voci "Il mio profilo" e "Prenota ora". Selezionando la prima opzione, viene visualizzato un sottomenù con le voci "Dati personali", "Le mie prenotazioni" e "Cartella clinica", come mostrato in Figura 5.6. Il menù a tendina è interagibile in tutte le schermate dedicate al paziente autenticato.

Figura 5.5: Vista relativa alla home page del paziente con menù a tendina

Figura 5.6: Vista relativa alla home page del paziente con sottomenù "Il mio profilo"

Dati personali

Selezionando la voce "Dati personali" dal menù a tendina, il paziente visualizza la schermata contenente le informazioni inserite in fase di registrazione, come mostrato in Figura 5.7.

Inoltre, egli può modificare la propria password selezionando l'opzione "Modifica password", la quale apre una sezione dedicata, come illustrato in Figura 5.8. Qualora l'utente desideri annullare la modifica, può farlo selezionando l'opzione "Annulla modifica password", che nasconderà la sezione di modifica.

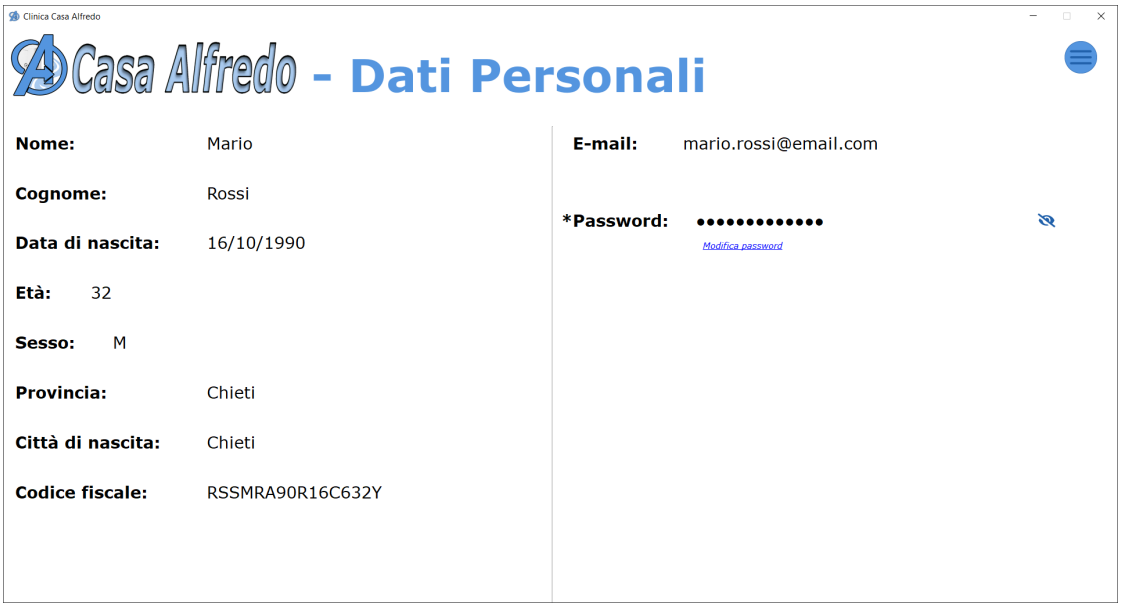

Figura 5.7: Vista relativa ai dati personali del paziente

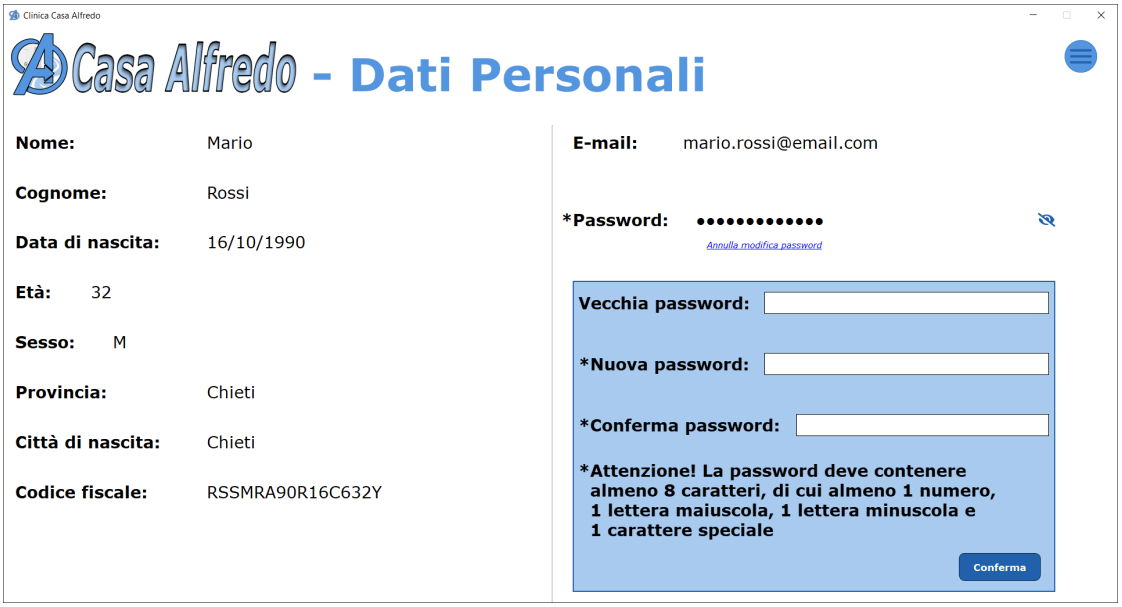

Figura 5.8: Vista relativa ai dati personali del paziente con sezione di modifica della password

Le mie prenotazioni

Selezionando la voce "Le mie prenotazioni" dal menù a tendina, il paziente visualizza la schermata che elenca le prenotazioni relative alle visite da effettuare nella clinica. Tale schermata è illustrata in Figura 5.9.

Alla fine di ogni record vi è un pulsante che permette la visualizzazione della relativa prenotazione in modo dettagliato, come mostrato in Figura 5.10.

Figura 5.9: Vista relativa alle prenotazioni del Paziente

Figura 5.10: Vista relativa ai dettagli della prenotazione

Da questa schermata è possibile tornare alla lista delle prenotazioni, selezionando l'opzione "Torna indietro", scaricare la prenotazione in formato PDF, tramite l'opzione "Download PDF", e, se lo si desidera, annullare la prenotazione effettuata, cancellando, così, la visita presso la clinica.

Cartella clinica

Selezionando la voce "Cartella clinica" dal menù a tendina, il paziente visualizza la schermata contenente le visite che ha effettuato presso la clinica. Ogni visita contiene una breve descrizione inserita dal personale medico operante. La schermata è illustrata in Figura 5.11.

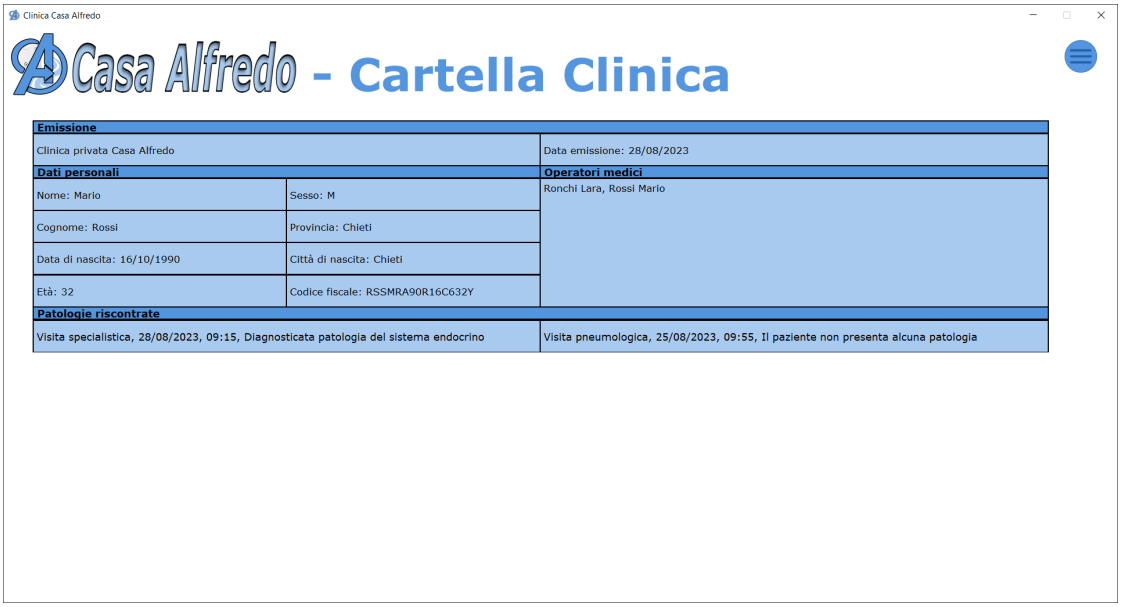

Figura 5.11: Vista relativa alla cartella clinica del paziente

Prenota ora

Selezionando la voce "Prenota ora" dal menù a tendina, il paziente visualizza una schermata che consente la prenotazione di una nuova visita. Egli viene guidato attraverso una serie di passi che gli facilitano la scelta della specifica attività ambulatoriale, del giorno e dell'ora in cui si vuol effettuare la visita. La schermata è illustrata in Figura 5.12.

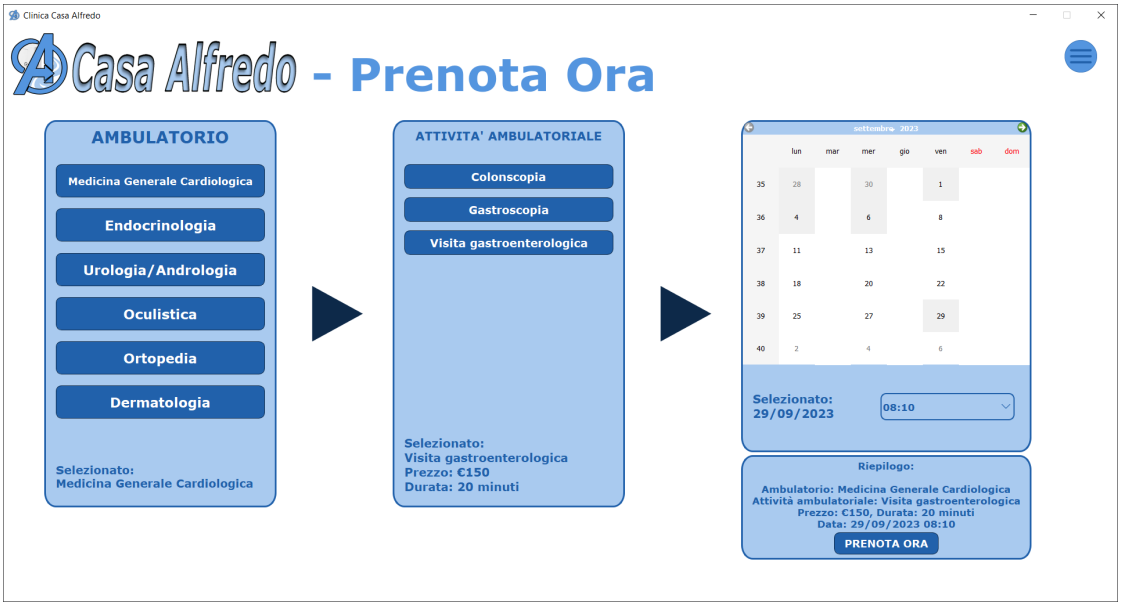

Figura 5.12: Vista relativa alla prenotazione di una visita

5.3.3 Sezione Personale medico

Come per il paziente, una volta effettuato l'accesso, un membro del personale medico può vedere la schermata in Figura 5.13. Da qui egli può interagire con il menù a tendina che

mostra le voci "Dati personali", "Lista prenotazioni" e "I miei pazienti". Il menù a tendina è interagibile in tutte le schermate del paziente autenticato.

Figura 5.13: Vista relativa alla home page del personale medico

Dati personali

Selezionando la voce "Dati personali" dal menù a tendina, un membro del personale medico visualizza la schermata contenente le proprie informazioni registrate nel sistema. Inoltre, come per il paziente, è possibile modificare la propria password selezionando l'opzione "Modifica password", la quale apre una sezione dedicata. La schermata è illustrata in Figura 5.14.

Figura 5.14: Vista relativa ai dati personali del personale medico

Lista prenotazioni

Selezionando la voce "Lista prenotazioni" dal menù a tendina, un membro del personale medico può vedere la schermata che elenca le prenotazioni in cui egli è coinvolto per svolgere

le proprie attività. La schermata è illustrata in Figura 5.16.

Figura 5.15: Vista relativa alla lista delle prenotazioni del personale medico

Alla fine di ogni record vi è un pulsante che permette la visualizzazione della cartella clinica del paziente associato alla specifica prenotazione, come illustrato in Figura 5.16. Dalla cartella clinica è possibile tornare alla lista delle prenotazioni del personale medico selezionando l'opzione "Torna indietro".

Figura 5.16: Vista relativa alla lista delle prenotazioni del personale medico

I miei pazienti

Selezionando la voce "I miei pazienti" dal menù a tendina, un membro del personale medico visualizza una schermata che presenta due sezioni distinte. Nella prima sezione viene elencata la lista delle visite già effettuate dai pazienti. In questo caso, un membro del personale medico ha la possibilità di aggiungere una descrizione alla visita, selezionando

l'opzione "Modifica". Una volta inserita la descrizione, la visita verrà collocata nella seconda sezione dell'interfaccia. La schermata contenente le due sezioni è illustrata in Figura 5.17.

Figura 5.17: Vista relativa alla lista dei pazienti visitati

CAPITOLO 6

SWOT Analysis

In questo capitolo presenteremo la SWOT Analysis, uno strumento che utilizzeremo per valutare il software per la clinica privata "Casa Alfredo". Successivamente identificheremo punti di forza, debolezze, opportunità e minacce. Questa analisi guiderà il miglioramento del software per fornire servizi più efficaci.

6.1 Introduzione

In questa sezione procederemo con la selezione dei componenti necessari per l'implementazione del software e della relativa base di dati, in riferimento alle informazioni descritte nei capitoli precedenti.

La SWOT Analysis è uno strumento essenziale nella pianificazione strategica, ampiamente utilizzato dalle aziende e dalle organizzazioni per valutare le caratteristiche di un progetto o, nel caso in questione, di un software.

Al fine di sviluppare una comprensione approfondita della situazione attuale e delle opportunità future, la SWOT Analysis coinvolge una valutazione critica e onesta di quattro aspetti:

- *Strengths* (Punti di forza);
- *Weaknesses* (Punti di debolezza);
- *Opportunities* (Opportunità);
- *Threats* (Minacce).

L'analisi può essere applicata sia all'ambiente interno, per esaminare punti di forza e di debolezza, che all'ambiente esterno, per valutare minacce ed opportunità. Una volta completata l'analisi SWOT, le aziende possono sfruttare queste informazioni per sviluppare strategie mirate al miglioramento del loro prodotto. Nella Figura 6.1 si può osservare una rappresentazione grafica di quanto descritto.

Nelle prossime sezioni verranno delineati i quattro aspetti della SWOT Analysis, relativi al software destinato alla clinica privata "Casa Alfredo".

Figura 6.1: Matrice SWOT

6.2 Strengths

Questa categoria riguarda gli aspetti interni positivi del software, ossia caratteristiche, risorse e competenze che lo rendono competitivo nel suo settore.

I punti di forza del software sono:

- *Automatizzazione dei processi*. Il software può automatizzare numerosi processi clinici, come la registrazione dei pazienti e la gestione delle visite, riducendo gli errori umani.
- *Alta efficienza*. Il software per la clinica privata "Casa Alfredo" è noto per velocità ed efficienza con cui esegue le operazioni: ciò lo rende molto attraente per gli utenti.
- *Accesso rapido alle informazioni*. Il software consente un accesso rapido e sicuro alle informazioni dei pazienti, come la visualizzazione della cartella clinica o delle prenotazioni.
- *Accesso remoto*. La possibilità di accesso da remoto consente ai medici di consultare le informazioni dei pazienti e di lavorare al di fuori della clinica, se necessario.
- *Interfaccia utente intuitiva*. L'interfaccia utente del software è stata progettata con cura per massimizzare la facilità d'uso, rendendo il software accessibile a un'ampia gamma di utenti.

6.3 Weaknesses

I punti di debolezza sono aspetti interni negativi, o limitazioni del software o del sistema, che richiedono miglioramenti o correzioni. Nel nostro caso essi sono:

- *Mancanza di alcune funzionalità chiave*. In confronto alla concorrenza, il sistema manca di alcune caratteristiche essenziali; un esempio rilevante è l'assenza di un'applicazione mobile che consentirebbe agli utenti di accedere al sistema in modo più conveniente. La mancanza di questa funzionalità limita anche la possibilità di ricevere eventuali notifiche sui propri dispositivi e di comunicare con i medici attraverso l'applicazione.

- *Dipendenza da terze parti*. Alcune sezioni del software sono dipendenti da componenti di terze parti, il che può comportare problemi di compatibilità e di sicurezza.
- *Resistenza al cambiamento da parte del personale medico*. È possibile che il personale medico mostri una certa resistenza nell'abbandonare i metodi tradizionali o i sistemi precedenti a favore del nuovo software. Questo atteggiamento può causare ritardi nella transizione e nella piena adozione del nuovo sistema.

6.4 Opportunities

Le opportunità rappresentano fattori esterni positivi che potrebbero essere sfruttati per il successo del software. In questo contesto abbiamo:

- *Espansione dei servizi*. Il software potrebbe consentire alla clinica di espandere la gamma di servizi offerti, come l'implementazione di un'applicazione mobile o la gestione delle prescrizioni online, per attirare un numero maggiore di pazienti.
- *Integrazione con dispositivi IoT*. L'integrazione del software con dispositivi IoT può consentire il monitoraggio in tempo reale dei pazienti e la raccolta di dati utili per la diagnosi e il trattamento.
- *Riduzione dei costi a lungo termine*. Nonostante i costi iniziali, il software potrebbe ridurre i costi operativi a lungo termine attraverso l'automazione e l'efficienza.
- *Nuove partnership strategiche*. L'azienda può cercare collaborazioni con altre aziende per ampliare la portata del software e offrire nuove funzionalità.

6.5 Threats

Le minacce rappresentano fattori esterni negativi o ostacoli che possono influire negativamente sul successo del software. Nel nostro caso abbiamo le seguenti minacce:

- *Normative in evoluzione*. Le normative governative in continua evoluzione potrebbero richiedere modifiche sostanziali al software per rimanere in conformità.
- *Rischio di sicurezza informatica*. Il software potrebbe essere vulnerabile a minacce informatiche, come hacking o furto di informazioni, mettendo a rischio la sicurezza dei dati dei pazienti.
- *Obsolescenza tecnologica*. Tecnologie più avanzate potrebbero rendere obsolete le funzionalità del software nel tempo, richiedendo costosi aggiornamenti o migrazioni.

6.6 Conclusione

In sintesi, l'analisi SWOT è uno strumento di grande utilità che guida l'azienda nella pianificazione strategica e nel processo decisionale. L'analisi del software destinato alla clinica privata "Casa Alfredo" ha fornito una visione dettagliata dei suoi punti di forza, delle debolezze, delle opportunità e delle minacce. Questa valutazione critica permette di comprendere appieno la situazione attuale del software e le potenziali direzioni strategiche per il futuro.

Utilizzando quest'analisi in modo proattivo, il software può continuare a migliorarsi, offrendo servizi medici sempre più efficaci e soddisfacendo le esigenze dei pazienti.

Conclusioni

Nel corso di questa tesi abbiamo esplorato approfonditamente il processo di sviluppo di un software dedicato alla gestione della clinica privata "Casa Alfredo".

Abbiamo iniziato il nostro viaggio esaminando le caratteristiche delle cliniche private, i vantaggi che presentano rispetto alle istituzioni pubbliche e le loro criticità, evidenziando il ruolo cruciale delle tecnologie ingegneristiche e informatiche nell'ottimizzare l'assistenza medica e l'esperienza del paziente.

Successivamente, ci siamo concentrati sulle fasi di specifica e di analisi dei requisiti del software, mettendo in luce l'importanza della definizione di requisiti funzionali e non funzionali.

Una volta accertata la completezza e la correttezza dei requisiti, abbiamo affrontato la progettazione della base di dati. Utilizzando il modello E-R, abbiamo descritto lo schema concettuale della base di dati per poi passare alla progettazione logica e, infine, alla traduzione dello schema concettuale nello schema logico, definendo tabelle e relazioni.

Per rappresentare al meglio l'infrastruttura del software, abbiamo utilizzato il linguaggio di modellazione UML, concentrandoci sui diagrammi delle classi e delle attività. Questa fase è cruciale per evitare problemi legati all'architettura del software durante l'implementazione. In particolare, durante il processo di progettazione, abbiamo dedicato un'attenzione significativa alla scelta del pattern architetturale più appropriato per il nostro sistema.

Dopo aver definito la struttura del software, si è passati alla sua implementazione, mettendo in evidenza le tecnologie utilizzate, e alla presentazione di un manuale utente, utile per orientare nell'uso dell'applicazione.

Infine, abbiamo condotto un'analisi SWOT per valutare il software sviluppato, identificando punti di forza, debolezze, opportunità e minacce.

Per concludere, è fondamentale considerare le potenziali direzioni per lo sviluppo futuro. Tra queste citiamo:

- *Integrazione di tecnologie avanzate*: si può pensare di esplorare l'integrazione di tecnologie emergenti, come l'Intelligenza Artificiale e l'apprendimento automatico, per migliorare la diagnosi medica e l'efficienza operativa.
- *Espansione delle funzionalità*: si può pensare di ampliare il software per includere nuove funzionalità che possano migliorare ulteriormente l'esperienza dei pazienti e semplificare la gestione della clinica, come il monitoraggio dei pazienti a distanza e l'introduzione di un sistema per il pagamento online delle prenotazioni.

- *Sviluppo di un'applicazione mobile*: si può pensare di creare un'applicazione mobile per consentire ai pazienti di accedere ai servizi della clinica e monitorare la propria salute in modo più agevole.
- *Sicurezza dei dati*: si può pensare di continuare a investire in robuste misure di sicurezza dei dati per garantire la privacy e la protezione delle informazioni dei pazienti.
- *Collaborazioni e partnership*: si può pensare di esplorare collaborazioni con altre cliniche private o fornitori di servizi medici per estendere la portata e la qualità dei servizi offerti.

Seguire queste direzioni per lo sviluppo futuro non solo migliorerebbe l'esperienza complessiva dei pazienti, ma potrebbe anche contribuire a mantenere la clinica all'avanguardia nel settore medico, garantendo una continua erogazione di servizi di alta qualità.

Bibliografia

SOMMERFIELD, I. (2017), *Introduzione all'ingegneria del software*, Pearson.

WIEGERS K. e HOKANSON, C. (2023), *Software Requirements Essentials: Core Practices for Successful Business Analysis*, Addison-Wesley Professional.

DAMIANI E. e MADRAVIO M. e BÖHM A. (2007), *UML pratico con elementi di ingegneria del software*, Pearson.

LARMAN, C. (2020), *Applicare UML e i pattern. Analisi e progettazione orientata agli oggetti*, Pearson.

ATZENI P. e CERI S., FRATERNALI P., PARABOSCHI S. e TORLONE R. (2023), *Basi di dati*, McGraw-Hill Education.

FERRARIO , M. L. (2016), *Tecniche di progettazione del software*, in riga edizioni.

SARSBY, A. (2016), *SWOT Analysis: A Guide to Swot for Business Studies Students*, Spectaris Ltd.

FITZPATRICK, P. (2016), *Create GUI Applications with Python & Qt5*, Stampato in proprio.

Siti web consultati

- Casa di Cura Quisisana – `https://www.clinicaquisisana.it`
- Clinica Privata Sanatrix – `https://www.clinicasanatrix.com`
- Clinica San Francesco – `https://www.clinicasanfrancesco.it`
- Clinica Santa Rita – `https://www.clinicasrita.it`
- Clinica Villa Margherita – `https://clinicavillamargherita.it`
- Dacrema.com – `http://www.dacrema.com/Informatica/gerarchie_elimin.htm`

- Devmy – `https://devmy.it/codevmy/portfolio/e-hcert-la-tua-cartella-clinica-a-portata-di-click`
- DidaWiki – `http://didawiki.di.unipi.it/lib/exe/fetch.php/informatica/is-a/is_08_19_attivitastati.pdf`
- Informatica e Ingegneria Online – `https://vitolavecchia.altervista.org`
- Innovatv – `https://www.innovatv.it/come-funziona-una-clinica-privata`
- Knowledge Systems Corporation – `https://www.cs.hmc.edu/~mike/courses/mike121/readings/reqsModeling/firesmith.htm`
- LinkedIn Pulse – `https://www.linkedin.com/pulse/mvc-mvcs-software-engineering-asher-toqeer`
- Personal Knowledge Base – `https://www.andreaminini.com`
- PAQ, Pubblica Amministrazione di Qualità – `http://qualitapa.gov.it/sitoarcheologico/relazioni-con-i-cittadini/utilizzare-gli-strumenti/analisi-swot/index.html`
- Quotidianosanità.it – `https://www.quotidianosanita.it/scienza-e-farmaci/articolo.php?articolo_id=102171`
- Stormshield – `https://www.stormshield.com/it/prodotti-e-soluzioni/per-settore-di-attivita/salute`
- Twproject – `https://twproject.com/blog/it/la-matrice-di-tracciabilita-cose-e-cosa-serve`
- Visure – `https://visuresolutions.com/it/blog/non-functional-requirements`
- Wikipedia – `www.wikipedia.org`

Ringraziamenti

Vorrei dedicare questa sezione a coloro che hanno contribuito in modo significativo al mio percorso di studi ad Ancona, periodo che ha segnato profondamente la mia vita.

Desidero ringraziare la mia famiglia, in particolare i miei genitori, per aver creduto in me, per avermi dato fiducia e supporto fin dal primo giorno.

La mia più profonda gratitudine va a Sara, la mia ragazza, costante fonte di motivazione, al mio fianco in ogni passo di questo viaggio. La sua comprensione e il suo appoggio nei momenti di stress hanno reso tutto più tollerabile; senza di lei questo traguardo non sarebbe stato possibile.

Tante sono le persone straordinarie che ho avuto il privilegio di conoscere durante il mio percorso universitario, tra le quali Kevin, che ringrazio per la sincera e preziosa amicizia durante questi ultimi anni.

Un ringraziamento speciale va al mio amico Rocco, compagno di avventure che ho conosciuto durante la quarantena; grazie per le nottate passate a chiacchierare, mangiare, giocare e programmare: conservo gelosamente questi ricordi.

Desidero rivolgere un pensiero particolare ai miei coinquilini, Daniel, Edoardo e Leonardo, che mi hanno accolto nella loro casa e nella loro vita. Con loro ho condiviso risate e momenti di crescita personale.

Inoltre, vorrei estendere la mia gratitudine ai cari amici di Lucera, che hanno seguito e sostenuto questo percorso anche a distanza.

Infine, desidero ringraziare il professor Domenico Ursino: nonostante la calura estiva, ha dimostrato una disponibilità ineguagliabile, rispondendo sempre alle mie richieste e guidandomi con saggezza e pazienza.

In conclusione, a tutti coloro che hanno reso questo percorso in qualche modo un'esperienza memorabile, compreso me stesso … Grazie di cuore.