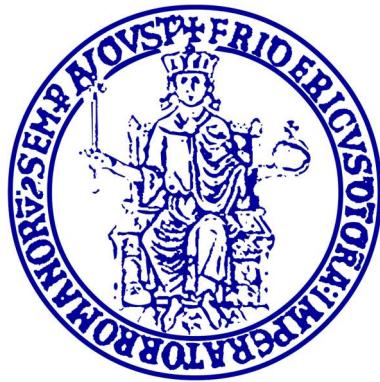


UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE DELL'INFORMAZIONE



CORSO DI LAUREA IN INFORMATICA

Tesi di Laurea Triennale

Shared Care Plan: una soluzione per la gestione di piani terapeutici condivisi

Tutor Accademico

Studente

Alessandro De Luca

Giuseppe Piccolo

N86002464

Tutor Aziendale

Stefano Tagliaferri

Anno accademico 2020/2021

SOMMARIO

Capitolo: 1 Introduzione.....	5
1.1 Scopo Tirocinio.....	5
1.2 Health Care.....	5
1.3 Digital health	6
1.4 Benefici del digital health	6
1.5 Ruolo progetto	7
1.6 Azienda sede Tirocinio	7
Capitolo: 2 Progettazione.....	8
2.1 Descrizione del progetto.....	8
2.2 Funzionalità	9
2.2.1 Funzionalità 1: Registrazione di un nuovo paziente	9
2.2.2 Funzionalità 2: Creazione/aggiornamento dello Shared Care Plan	9
2.2.3 Funzionalità 3: Visualizza lo storico di parametri misurati e obiettivi precedentemente registrati.....	10
2.2.4 Funzionalità 4: Visualizza lo storico delle prescrizioni e somministrazioni.....	10
2.2.5 Funzionalità 5: Registrazione dell'avvenuto monitoraggio.....	10
2.2.6 Funzionalità 6: Registrazione dell'avvenuta somministrazione della terapia.....	11
2.3 Modellazione casi d'uso.....	12
2.4 Tabelle di cockburn per ogni caso d'uso.....	13
2.5 Mockup	27
2.6 Modello di dominio.....	34
2.7 Diagrammi di sequenza di analisi	41
2.8 Diagrammi di stato di analisi.....	47
2.9 Diagrammi di attività	50
Capitolo: 3 Implementazione e tecnologie	53
3.1 Architettura client-server	53
3.2 Comunicazione http	53
3.3 JSON	54
3.4 REST API	54
3.5 Sicurezza e autenticazione.....	55

3.5.1	Hashing password.....	55
3.5.2	Funzionamento hashing	56
3.5.3	Autenticazione utente.....	56
3.5.4	Funzionamento autenticazione.....	56
3.5.5	Middleware	57
3.5.6	Logout.....	57
3.6	Paradigmi di programmazione	58
3.6.1	Lato server	58
3.6.2	Lato client	58
3.7	DOM e Virtual DOM	58
3.7.1	DOM.....	58
3.7.2	Virtual DOM.....	59
3.8	Ciclo di vita dei componenti.....	59
3.8.1	Mounting	59
3.8.2	Updating	60
3.8.3	Unmounting.....	60
3.9	React hook	60
3.10	Reactjs Design Pattern	61
3.10.1	Funzioni stateless	61
3.10.2	Conditional rendering.....	61
3.10.3	Controlled components.....	61
3.11	Routing in React	62
3.11.1	Routers	62
3.11.2	History	62
3.11.3	Routes.....	63
3.11.4	Switch	63
3.11.5	Link.....	64
3.12	Express.js	64
3.12.1	Caratteristiche di Express.js.....	65
3.12.2	Vantaggi di Express.js	65
3.12.3	Routing in Express.js.....	66
3.13	Webpack.....	67
3.14	Fetching	69
3.14.1	Funzione fetch	69

3.14.2	Axios.....	69
3.15	Tecnologie.....	70
3.15.1	Javascript.....	70
3.15.2	TypeScript.....	70
3.15.3	ReactJS	71
3.15.4	JSX	71
3.16	71
3.16.1	Material UI	71
3.16.2	CSS.....	71
3.16.3	NodeJS.....	72
3.16.4	Express	72
3.16.5	AXIoS	72
3.16.6	Npm.....	72
3.16.7	PostgreSQL.....	72
3.16.8	Visual Studio Code	73
3.16.9	Git.....	73
3.16.10	Gitlab.....	73
Capitolo: 4	Testing.....	74
4.1	Descrizione attività	74
4.2	Testing del sistema.....	74
Capitolo: 5	Conclusioni.....	80
5.1	Sviluppi futuri	81
5.1.1	Funzionalità aggiuntive	81
5.1.2	Riadattamento per altri dispositivi	81
5.1.3	Sviluppo nativo per Android e IOS.....	82

CAPITOLO: 1 INTRODUZIONE

1.1 SCOPO TIROCINIO

Lo scopo del tirocinio è stato quello di sviluppare un prototipo di uno Shared Care Plan. Uno Shared Care Plan è una piattaforma in cui gli utenti autorizzati, possono gestire e monitorare dati e informazioni condivise da altri utenti. In questo caso specifico, il Care Plan è stato sviluppato per l'ambito medico, in modo tale da permettere al personale come medici e infermieri, di monitorare e assistere lo stato di salute di un paziente fragile, rendendo la comunicazione e la cooperazione.

Oggi giorno grazie alle reti internet sempre più veloci, a browser che diventano sempre più performanti e le tecnologie usate Web che migliorano nel tempo, è possibile sviluppare applicativi software che invece di essere eseguiti dal sistema operativo, sono interpretati ed eseguiti dal browser, senza avere enormi perdite di prestazioni. Questo permette di avere massima compatibilità sui vari sistemi operativi e sui vari dispositivi, visto che un tale applicativo ha solo bisogno di un browser per funzionare. Non si avrà quindi il bisogno di sviluppare varie versioni dello stesso applicativo per varie piattaforme, riducendo costi e tempo di sviluppo. Anche la manutenibilità e l'aggiornamento dell'applicativo sono più rapidi e facili da eseguire. Quindi la web application potrà essere raggiunta facilmente da chiunque e dappertutto nel mondo e l'utente finale avrà un servizio sempre funzionante, sempre aggiornato e avrà sempre accesso alle ultime informazioni dell'applicativo.

L'attività di tirocinio si è quindi formalizzata nella progettazione e sviluppo di un prototipo di uno Shared Care Plan, utilizzando strumenti che permettono la realizzazione di Web Application.

1.2 HEALTH CARE

L'health care, o assistenza sanitaria, è il mantenimento o il miglioramento della sanità attraverso la prevenzione, diagnosi, trattamento, recupero o cura di malattie, patologie, infortuni e altre menomazioni fisiche e mentali di persone. L'health care è fornito da professionisti della sanità che lavorano nei campi della: medicina, odontoiatria, farmacia, psicologia, assistenza infermieristica, preparazione atletica e altri.

1.3 DIGITAL HEALTH

L'ampio spettro del digital health, o assistenza sanitaria digitale, include categorie come: mobile health, health information technology, dispositivi indossabili e telemedicina. Grazie alle applicazioni mediche per mobile e software che supportano le decisioni cliniche dei dottori e grazie a software che utilizzano sempre di più strumenti del campo dell'intelligenza artificiale e del campo del machine learning, esse ci stanno guidando nella rivoluzione dell'assistenza sanitaria. Questi strumenti hanno un grande potenziale per migliorare le abilità per diagnosticare e trattare patologie e di fornire l'assistenza sanitaria per ogni essere umano. Le tecnologie del digital health usano computing platform, connettività, software e sensori per l'health care e gli utenti correlati. Queste tecnologie si estendono in vari casi d'uso, dalle applicazioni per il benessere in generale, alle applicazioni per dispositivi medici. Esse possono anche essere usate per lo sviluppo o lo studio di prodotti medici.

1.4 BENEFICI DEL DIGITAL HEALTH

Gli strumenti digitali stanno dando al personale medico un punto di vista sempre più olistico della salute del paziente attraverso l'accesso a dati e dando al paziente un controllo più accurato sulla propria salute. Il digital health offre opportunità reali per migliorare esiti medicali e migliorare l'efficienza. Queste tecnologie possono responsabilizzare i consumatori a fare decisioni migliori riguardo alla loro salute e fornire nuove opzioni per facilitare: la prevenzione, diagnosi anticipate di patologie che mettono a rischio la vita del paziente e gestione di condizioni croniche al di fuori dei metodi tradizionali dell'assistenza sanitaria. Quindi le tecnologie per il digital health vengono usati per:

- ridurre inefficienze;
- migliorare l'accesso all'health care;
- ridurre i costi;
- migliorare la qualità;
- realizzare medicine più personalizzate per i pazienti.

L'uso delle tecnologie, come smartphone, social network e web application, non solo stanno cambiando il modo di comunicare, ma forniscono anche modi innovativi per noi per monitorare la nostra salute e il nostro benessere e per darci un maggiore accesso delle informazioni. Assieme, questi avanzamenti stanno portando a una convergenza di persone, informazioni, tecnologie e connettività per migliorare l'health care e i suoi risultati.

1.5 RUOLO PROGETTO

Il mio ruolo durante la fase di sviluppo del progetto è stato quello di:

- progettare un database che permette la memorizzazione dei dati necessari per il corretto funzionamento dello Shared Care Plan;
- progettare un server che permette di accontentare le richieste effettuate da ogni client, garantendo un certo livello di sicurezza e garantendo buoni livelli di prestazione;
- progettare un client con una interfaccia utente grafica funzionale che permette all'utente di utilizzare le caratteristiche fornite della web app nel modo più intuitivo possibile.

1.6 AZIENDA SEDE TIROCINIO

Kelyon è un'azienda fondata nel 2008 che si occupa di Digital Health, che ha come obiettivo quello di digitalizzare i processi così da cercare di fornire migliore supporto di assistenza sanitaria agli esperti di questo settore per la gestione, prevenzione, diagnosi e trattamento dei loro pazienti che soffrono di malattie complesse.

CAPITOLO: 2 PROGETTAZIONE

2.1 DESCRIZIONE DEL PROGETTO

Lo **Shared Care Plan** (SCP) rappresenta un documento condiviso tra più professionisti (es. personale medico, infermieri, farmacisti, OSS, nutrizionisti, fisioterapisti, ecc.) per gestire in maniera olistica e collaborativa un paziente fragile o la cui condizione richiede un approccio multidisciplinare. Lo Shared Care Plan crea una rete di assistenza attorno al paziente, includendolo attivamente nel suo percorso di cura (eventualmente includendo il suo caregiver) e rendendo la comunicazione e la cooperazione tra il team multidisciplinare e il paziente più efficace ed efficiente. Lo Shared Care Plan rappresenta il punto in cui vengono definito l'intero percorso di cura multidimensionale e integrato (es. comprende il piano nutrizionale, il piano di attività fisica, i piani terapeutici, ecc.) per il raggiungimento degli obiettivi di salute, personalizzati in base alle necessità del singolo paziente.

L'obiettivo del progetto consiste nello sviluppo di una versione semplificata dello Shared Care Plan, attraverso lo sviluppo di un prototipo di piattaforma utilizzata da:

- **Medico (Care plan manager)**: utilizza la web app per registrare nuovi pazienti, accedere successivamente ai loro dati, monitorare il loro stato di salute, definire nuovi obiettivi e revisionare l'intero SCP;
- **Infermiere (Care team member)**: utilizza la web app per accedere ai dati dei pazienti, dei loro SCP personalizzati, registrare le diverse misurazioni/attività eseguite, secondo quanto indicato dal medico.

2.2 FUNZIONALITÀ

2.2.1 FUNZIONALITÀ 1: REGISTRAZIONE DI UN NUOVO PAZIENTE

Attore principale: Medico.

- la web app mostra all'utente l'intero elenco di pazienti precedentemente registrati e il pulsante per registrare un nuovo paziente;
- l'utente clicca il pulsante "Nuovo paziente"
- la web app mostra all'utente il form di registrazione di un nuovo paziente, in cui si richiedono nome, cognome, data di nascita, sesso, codice fiscale, comune e provincia di nascita, comune e provincia di residenza
- la web app richiede all'utente i dati anamnestici del paziente (peso [kg], altezza [cm], allergie, intolleranze, patologie pregresse, interventi chirurgici, vaccinazioni)
- a seguito dell'inserimento di peso e altezza, la web app mostra all'utente il BMI attuale calcolato mediante la formula $BMI = \text{peso in kg} / \text{altezza in m}^2$
- Per ogni paziente, l'utente può visualizzare i dati, il care plan, registrare nuove misurazioni/attività
- Dopo aver aggiunto i dati, l'utente preme "Salva" per aggiungere il nuovo paziente.

2.2.2 FUNZIONALITÀ 2: CREAZIONE/AGGIORNAMENTO DELLO SHARED CARE PLAN

Attore principale: Medico.

- dall'elenco di pazienti, l'utente avvia la creazione di uno shared care plan
- la web app mostra i dati personali e anamnestici precedentemente inseriti
- l'utente inserisce la data di prima compilazione dello Shared Care Plan e la periodicità con cui andrà revisionato (1 volta al mese, 1 volta ogni 6 mesi, ecc.)
- l'utente inserisce ulteriori parametri del paziente (temperatura [$^{\circ}\text{C}$], SpO₂ [%], frequenza cardiaca [bpm], pressione sistolica e diastolica [mmHg])
- per ogni parametro clinico modificabile (peso, SpO₂, frequenza cardiaca, pressione sanguigna) relativo al paziente, l'utente può definire un obiettivo (es. pressione sistolica < 130 mmHg e peso < 80 kg) che dovrebbe essere raggiunto entro la prossima revisione dello Shared Care Plan
- l'utente seleziona quali parametri sono da monitorare per lo specifico paziente durante il periodo di validità dello Shared Care Plan
- l'utente può prescrivere una o più terapie farmacologiche

- Per ogni prescrizione, l'utente deve specificare il principio attivo, modalità di somministrazione, dosaggio, frequenza di somministrazione (es. 1 compressa al giorno) e durata del trattamento

2.2.3 FUNZIONALITÀ 3: VISUALIZZA LO STORICO DI PARAMETRI MISURATI E OBIETTIVI PRECEDENTEMENTE REGISTRATI

- Attori principali: Medico o infermiere.
- dal Care Plan di un determinato paziente l'utente può visualizzare lo storico dei parametri misurati e precedentemente registrati
- l'utente clicca sul pulsante "Monitoraggio"
- la web app mostra l'elenco di parametri da monitorare e le misurazioni effettuate in forma tabellare e grafica
- l'utente può selezionare uno specifico parametro in modo tale da vedere le misurazioni effettuate, la periodicità, il valore obiettivo definito precedentemente ed eventuali note

2.2.4 FUNZIONALITÀ 4: VISUALIZZA LO STORICO DELLE PRESCRIZIONI E SOMMINISTRAZIONI

Attori principali: Medico o infermiere.

- Attori principali: Medico o infermiere.
- dal Care Plan di un determinato paziente l'utente può visualizzare lo storico delle prescrizioni e somministrazioni
- l'utente clicca sul pulsante "Terapie"
- la web app mostra l'elenco di terapie farmacologiche prescritte e somministrazioni effettuate in forma tabellare
- l'utente può selezionare una specifica terapia farmacologica in modo tale da vedere le somministrazioni effettuate, la periodicità, la durata della terapia ed eventuali note

2.2.5 FUNZIONALITÀ 5: REGISTRAZIONE DELL'AVVENUTO MONITORAGGIO

Attori principali: Medico o infermiere.

- per ogni paziente, l'utente può registrare una misurazione del/dei parametro/i da misurare secondo quanto stabilito nello Shared Care Plan

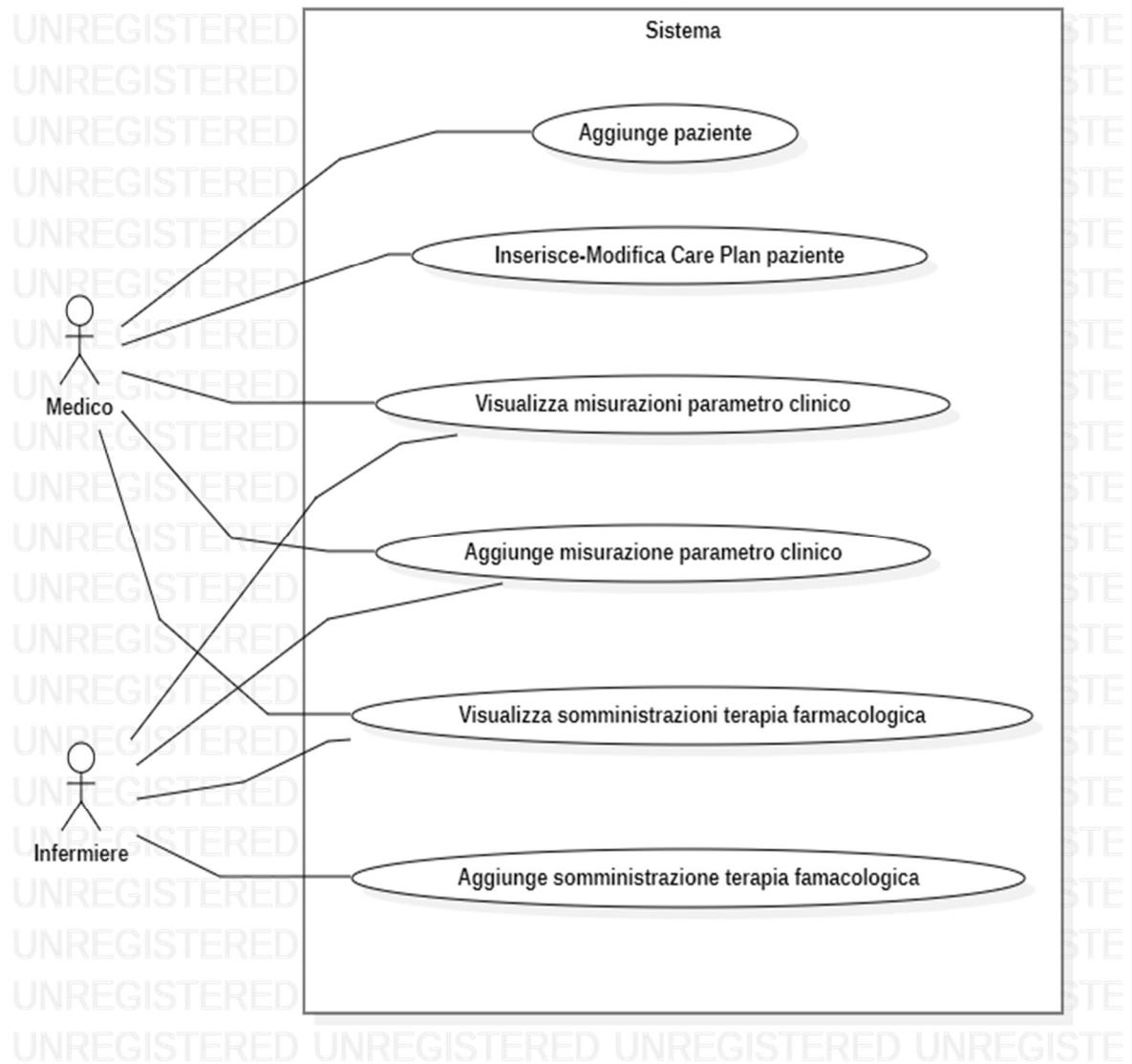
- dall'elenco delle misurazioni l'utente preme "Nuova misurazione"
- la web app mostra una finestra in cui l'utente aggiunge una misurazione al parametro selezionato
- per ogni misurazione, l'utente deve indicare data e ora dell'avvenuta misurazione, il valore misurato ed eventuali note

2.2.6 FUNZIONALITÀ 6: REGISTRAZIONE DELL'AVVENUTA SOMMINISTRAZIONE DELLA TERAPIA

Attore principale: infermiere.

- per ogni paziente, l'utente può registrare l'avvenuta somministrazione della terapia prescritta secondo quanto stabilito nello Shared Care Plan
- dall'elenco delle somministrazioni l'utente preme "Registra somministrazione"
- la web app mostra una finestra in cui l'utente aggiunge una nuova somministrazione alla terapia farmacologica selezionata
- per ogni somministrazione, l'utente deve indicare data e ora della somministrazione, se la somministrazione è avvenuta con successo ed eventuali note

2.3 MODELLAZIONE CASI D'USO



2.4 TABELLE DI COCKBURN PER OGNI CASO D'USO

Use Case #1	Aggiunge paziente		
Goal in Context	Il medico può aggiungere un nuovo paziente alla web application inserendo i dati di quest'ultimo		
Preconditions	Il medico deve aver effettuato l'accesso		
Success End Condition	Il medico aggiunge con successo un nuovo paziente		
Failed End Condition	Il medico visualizza un messaggio d'errore		
Primary Actor	Medico		
Trigger	L'utente preme "Nuovo paziente" in "lista pazienti"		
Description	Step n°	Medico	Sistema
	1		Mostra "aggiungi paziente"
	2	Inserisce i dati del paziente	
	3	Preme "Salva"	
	4		Mostra "messaggio dialogbox" con messaggio "Paziente aggiunto con successo"
	5	Preme "Ok"	
	6		Mostra "aggiungi-modifica care plan"

Extensions	Step n°	Medico	Sistema
Extension #1 Uno o più campi vuoti	2,1	Preme "Salva"	
	2,2		Mostra messaggio di errore e termina caso d'uso
Extension #2 Codice fiscale già esistente	4,1		Mostra messaggio di errore e termina caso d'uso
Extension #3 Connessione assente	4,1		Mostra messaggio di errore e termina caso d'uso
Extension #4 Annulla aggiunta paziente	3,1	Preme "Annulla"	
	3,2		Mostra "lista pazienti" e termina il caso d'uso
Extension #5 Esegue logout	2,1	Preme "Logout"	
	2,2		Mostra "conferma dialogbox"
	2,3	Preme "Si"	
	2,4		Mostra "pagina di benvenuto"
Extension #6 Non esegue logout	2,1	Preme "Logout"	
	2,2		Mostra "conferma dialogbox"
	2,3	Preme "No"	
	2,4		Torna al passo 1 dello scenario principale
Subvariations	Step n°	Medico	Sistema
Subvariation #1 Aggiunge paziente senza interventi chirurgici	2,1	Inserisce i dati del paziente ma senza interventi chirurgici	
	2,2	Preme "Salva"	
	2,3		Torna al passo 4 dello scenario principale

Use Case #2	Inserisce/Modifica care plan		
Goal in Context	Il medico può inserire o modificare il care plan di un determinato paziente		
Preconditions	Il medico deve aver effettuato l'accesso		
Success End Condition	Il medico aggiunge/modifica con successo il care plan di un paziente		
Failed End Condition	Il medico visualizza un messaggio d'errore		
Primary Actor	Medico		
Trigger	Preme "Aggiungi/Aggiorna care plan" a un determinato paziente in "lista pazienti medico"		
Description	Step n°	Medico	Sistema
	1		Mostra "inserisci-modifica care plan"
	2	Inserisce dati del care plan	
	3	Inserisce valori parametri clinici	
	4	Selezione parametri da monitorare	
	5	Selezione frequenza parametri da monitorare	
	6	Aggiunge terapie farmacologiche	
	7	Preme "Salva"	
	8		Mostra "messaggio dialogbox" con messaggio "Care plan aggiunto (modificato) con successo"

Extensions	Step n°	Medico	Sistema
Extension #1 Uno o più campi vuoti	2,1	Preme "Salva"	
	2,2		Mostra messaggio di errore e termina caso d'uso
Extension #2 Campi terapia farmacologica senza dati	8,1		Mostra messaggio di errore e termina caso d'uso
Extension #3 Connessione assente	8,1		Mostra messaggio di errore e termina caso d'uso
Extension #4 Annulla inserimento/modifica care plan	7,1	Preme "Indietro"	
	7,2		Mostra "lista pazienti" e termina il caso d'uso
Extension #5 Esegue logout	2,1	Preme "logout"	
	2,2		Mostra "conferma dialogbox"
	2,3	Preme "Si"	
	2,4		Mostra "pagina di benvenuto" e termina caso d'uso
Extension #5 Non esegue logout	2,1	Preme "logout"	
	2,2		Mostra "conferma dialogbox"
	2,3	Preme "No"	
	2,4		Torna al passo 1 dello scenario principale

Subvariations	Step n°	Medico	Sistema
Subvariation #1 Nessuna terapia farmacologica	7,1	Preme "Salva"	
	7,2		Torna al passo 8 dello scenario principale
Subvariation #2 Nessun parametro da monitorare	4,1	Aggiunge terapie farmacologiche	
	4,2	Preme "Salva"	
	4,3		Torna al passo 8 dello scenario principale
Subvariation #3 Rimuove una o più terapie farmacologiche	6,1	Rimuove terapie farmacologiche	
	6,2	Preme "Salva"	
	6,3		Torna al passo 8 dello scenario principale

Use Case #3	Visualizza misurazioni parametri clinici		
Goal in Context	Il medico o l'infermiere visualizza tutte le misurazioni che sono state effettuate di un parametro clinico a un determinato paziente		
Preconditions	Il medico o l'infermiere deve aver effettuato l'accesso		
Success End Condition	Il medico o infermiere visualizza con successo tutte le misurazioni di un parametro clinico		
Failed End Condition	Il medico o l'infermiere visualizza un messaggio d'errore		
Primary Actor	Medico, Infermiere		
Trigger	Preme "Monitora" in "aggiungi-modifica care plan" o in "visualizza care plan"		
Description	Step n°	Medico, Infermiere	Sistema
	1		Mostra "Visualizza misurazioni parametri clinici"
	2	Seleziona parametro	
	3		Mostra misurazioni parametro

Extensions	Step n°	Medico, Infermiere	Sistema
Extension #1 Nessun parametro da monitorare	1,1		Mostra messaggio di errore e termina caso d'uso
Extension #2 Connessione assente	1,1		Mostra messaggio di errore e termina caso d'uso
Extension #3 Esegue logout	2,1	Preme "logout"	
	2,2		Mostra "conferma dialogbox"
	2,3	Preme "Si"	
	2,4		Mostra "pagina di benvenuto" e termina caso d'uso
Extension #4 Non esegue logout	2,1	Preme "logout"	
	2,2		Mostra "conferma dialogbox"
	2,3	Preme "No"	
	2,4		Torna al passo 1 dello scenario principale
Extension #5 Torna indietro	2,1	Preme "Indietro"	
	2,2		Mostra "aggiungi-modifica care plan" se è un medico o mostra "visualizza care plan" se è infermiere e termina caso d'uso

Use Case #4	Aggiunge misurazione parametro clinico		
Goal in Context	Il medico o l'infermiere aggiunge una misurazione di un parametro clinico di un determinato paziente		
Preconditions	Il medico o l'infermiere deve aver effettuato l'accesso		
Success End Condition	Il medico o infermiere aggiunge con successo una nuova misurazione di un parametro clinico		
Failed End Condition	Il medico o l'infermiere visualizza un messaggio d'errore		
Primary Actor	Medico, Infermiere		
Trigger	Preme "Nuova misurazione" in "visualizza misurazioni parametri clinici"		
Description	Step n°	Medico, Infermiere	Sistema
	1		Mostra "Nuova misurazione"
	2	Inserisce valore misurato	
	3	Inserisce data misurazione	
	4	Inserisce nota	
	5	Preme "Salva"	
	6		Mostra "messaggio dialogbox" con messaggio "Misurazione aggiunta"

Extensions	Step n°	Medico, Infermiere	Sistema
Extension #1 Nessuna data	3,1	Inserisce nota	
	3,2	Preme "Salva"	
	3,3		Mostra messaggio di errore e termina caso d'uso
Extension #2 Connessione assente	6,1		Mostra messaggio di errore e termina caso d'uso
Extension #3 Nessun valore misurato	2,1	Inserisce data misurazione	
	2,2	Inserisce nota	
	2,3	Preme "Salva"	
	2,4		Mostra messaggio di errore e termina caso d'uso
Extension #4 Torna indietro	5,1	Preme "Indietro"	
	5,2		Mostra "visualizza misurazioni parametri clinici" e termina caso d'uso
Subvatiations	Step n°	Medico, Infermiere	Sistema
Subvatiations #1 Nessuna nota	4,1	Preme "Salva"	
	4,2		Torna al passo 6 dello scenario principale

Use Case #5	Visualizza somministrazioni terapia farmacologica		
Goal in Context	Il medico o l'infermiere visualizza le somministrazioni effettuate a un paziente di una determinata terapia farmacologica		
Preconditions	Il medico o l'infermiere deve aver effettuato l'accesso		
Success End Condition	Il medico o l'infermiere visualizza le somministrazioni effettuate		
Failed End Condition	Il medico o l'infermiere visualizza un messaggio d'errore		
Primary Actor	Medico, Infermiere		
Trigger	Preme "Terapie" in "aggiungi-modifica care plan"		
Description	Step n°	Medico, Infermiere	Sistema
	1		Mostra "visualizza somministrazioni terapia"
	2	Selezione terapia	
	3		Mostra somministrazioni

Extensions	Step n°	Medico, Infermiere	Sistema
Extension #1 Connessione assente	1,1		Mostra messaggio di errore e termina caso d'uso
Extension #2 Nessuna terapia esistente	1,1		Mostra messaggio di errore e termina caso d'uso
Extension #3 Esegue logout	2,1	Preme "logout"	
	2,2		Mostra "conferma dialogbox"
	2,3	Preme "Si"	
	2,4		Mostra "pagina di benvenuto" e termina il caso d'uso
Extension #4 Non esegue il logout	2,1	Preme "logout"	
	2,2		Mostra "conferma dialogbox"
	2,3	Preme "No"	
	2,4		Torna al passo 1 dello scenario principale

Use Case #6	Aggiunge somministrazione terapia famacologica		
Goal in Context	L'infermiere aggiunge una somministrazione a una determinata terapia farmacologica		
Preconditions	L'infermiere deve aver effettuato l'accesso		
Success End Condition	L'infermiere registra una somministrazione con successo		
Failed End Condition	L'infermiere visualizza un messaggio d'errore		
Primary Actor	Infermiere		
Trigger	Preme "Registra somministrazione" in "visualizza somministrazioni terapia"		
Description	Step n°	Infermiere	Sistema
	1		Mostra "aggiungi somministrazione"
	2	Inserisce data di somministrazione	
	3	Spunta la checkbox	
	4	Inserisce nota	
	5	Preme "Salva"	
	6		Mostra "messaggio dialogbox" con messaggio "Somministrazione aggiunta con successo"

Extensions	Step n°	Infermiere	Sistema
Extension #1 Connessione assente	6,1		Mostra messaggio di errore e termina caso d'uso
Extension #2 Nessuna data di somministrazione	2,1	Spunta la checkbox	
	2,2	Inserisce nota	
	2,3	Preme "Salva"	
	2,4		Mostra messaggio di errore e termina caso d'uso
Extension #3 Torna indietro	5,1	Preme "Indietro"	
	5,2		Mostra "visualizza somministrazioni terapia" e termina caso d'uso

Subvariations	Step n°	Infermiere	Sistema
Subvariation #1 Nessuna spunta al checkbox	3,1	Inserisce nota	
	3,2	Preme "Salva"	
	3,3		Torna al passo 6 dello scenario principale
Subvariation #2 Nessuna nota	4,1	Preme "Salva"	
	4,2		Mostra "messaggio dialogbox" con messaggio "Somministrazione" aggiunta con successo

2.5 MOCKUP

Listo pazienti

A Web Page

John Doe
Medico

> Elenco pazienti

NUOVO PAZIENTE

Paziente	data di nascita	codice fiscale	
Giovanni Rossi	11/03/1950	CNFQVS35D59Z355O	Visualizza dati paziente
Greta Bergamaschi	12/12/1958	DMBCYS37E63D243O	Crea il Care Plan
Elia Bruno	13/10/1948	NCULYT71C55E351T	Aggiorna il Care Plan
Alma Baresi	09/08/1944	GXKBFC38T12I750S	
Edmondo Moretti	07/07/1940	FJSMRV39H25F563Y	
Lilla Folliero	19/08/1950	VYYWLH75T09L302P	
Vittorio Piccio	20/12/1950	FFBRNV30E41H873G	
Gofreddo Folliero	10/02/1947	XDNQFJ72T11E810P	

Aggiungi paziente

A Web Page

http://

John Doe
Medico

> Aggiungi paziente

Dati paziente

Nome	Cognome	Data di nascita
<input type="text"/>	<input type="text"/>	<input type="text"/> gg / mm / aaaa 
Codice fiscale	Comune di nascita	Provincia di nascita
<input type="text"/>	<input type="text"/>	<input type="text"/>
Comune di residenza	Provincia di nascita	
<input type="text"/>	<input type="text"/>	

Dati anamnestici

Peso (kg)	Altezza (cm)	BMI (kg/m ²)
<input type="text"/>	<input type="text"/>	<input type="text"/>

Allergie

- arachidi
- soia
- molluschi
- acari della polvere
- nichel

Patologie pregresse

- Ipertensione arteriosa
- Diabete mellito di tipo II
- Demenza
- BPCO
- Alzheimer
- Parkinson

Vaccinazioni

- Covid-19
- Influenza
- Antitetanica
- Meningiti
- Epatite di tipo B
- Vaccino trivalente MPR
- Vaccino esavalente
- Anti HPV

Interventi chirurgici

SALVA

Aggiungi-Modifica care plan

A Web Page

http://

John Doe
Medico

> GIOVANNI ROSSI

DATI PAZIENTE CARE PLAN

Care Plan

Data compilazione

Periodicità di revisione

Parametri vitali

Temperatura (°C)
Valore misurato
Valore obiettivo Monitora

SpO2 (%)
Valore misurato
Valore obiettivo Monitora Periodicità

Frequenza cardiaca (bpm)
Valore misurato
Valore obiettivo Monitora

Pressione sistolica (mmHg) Pressione diastolica (mmHg)
Valore misurato Valore obiettivo Monitora

Peso (Kg)
Valore misurato
Valore obiettivo Monitora

Terapia farmacologica

Principio attivo Dosaggio
Modalità di somministrazione Frequenza di somministrazione
Durata del trattamento

Aggiungi nuova terapia

SALVA

Visualizza misurazioni parametri clinici

A Web Page

← → ⌂ Q https://

Jennifer Winston
Infermiere

> GIOVANNI ROSSI

DATI PAZIENTE CARE PLAN MONITORAGGIO TERAPIA

MONITORAGGIO

PESO Obiettivo attuale: 100 kg Frequenza monitoraggio: 1/mese NUOVA MISURAZIONE

Data misurazione	Valore misurato (kg)	Valore obiettivo (kg)	note
11/03/2021	100	100	
11/02/2021	110	100	
11/01/2021	120	115	
11/12/2020	130	120	
11/11/2020	140	120	

The graph displays the following approximate data points from the table:

Data misurazione	Valore misurato (kg)
11/03/2021	100
11/02/2021	110
11/01/2021	120
11/12/2020	130
11/11/2020	140

Nuova misurazione

A Web Page

https://

Jennifer Winston
Infermiere

> GIOVANNI ROSSI

DATI PAZIENTE CARE PLAN MONITORAGGIO TERAPIA

MONITORAGGIO

PESO Obiettivo attuale: 100 kg Frequenza monitoraggio: 1/mese NUOVA MISURAZIONE

Data misurazione	Valore misurato (kg)
11/03/2021	100
11/02/2021	110
11/01/2021	120
11/12/2020	130
11/11/2020	140

Data misurazione Ora misurazione
11 10 30
Valore misurato
Note
SAVE

The graph displays the following approximate data points from the table:

Data misurazione	Valore misurato (kg)
11/03/2021	100
11/02/2021	110
11/01/2021	120
11/12/2020	130
11/11/2020	140

Visualizza somministrazioni terapie

A Web Page

http://

Jennifer Winston
Infermiere

> GIOVANNI ROSSI

DATI PAZIENTE CARE PLAN MONITORAGGIO TERAPIA

TERAPIA FARMACOLOGICA

Amoxicillina ▾

Dosaggio: 1 compressa Modalità di somministrazione: orale
Frequenza: 1 volta al giorno Durata: 10 giorni

REGISTRA SOMMINISTRAZIONE

Data e ora	somministrato	note
11/03/2021	Sì	
11/02/2021	No	

Aggiungi somministrazione

A Web Page

http://

Jennifer Winston
Infermiere

> GIOVANNI ROSSI

DATI PAZIENTE CARE PLAN MONITORAGGIO TERAPIA

TERAPIA FARMACOLOGICA

Amoxicillina ▾

Dosaggio: 1 compressa Modalità di somministrazione: orale
Frequenza: 1 volta al giorno Durata: 10 giorni

REGISTRA SOMMINISTRAZIONE

Data e ora	somministrato
11/03/2021	Si
11/02/2021	No

Data somministrazione Ora somministrazione
 / /

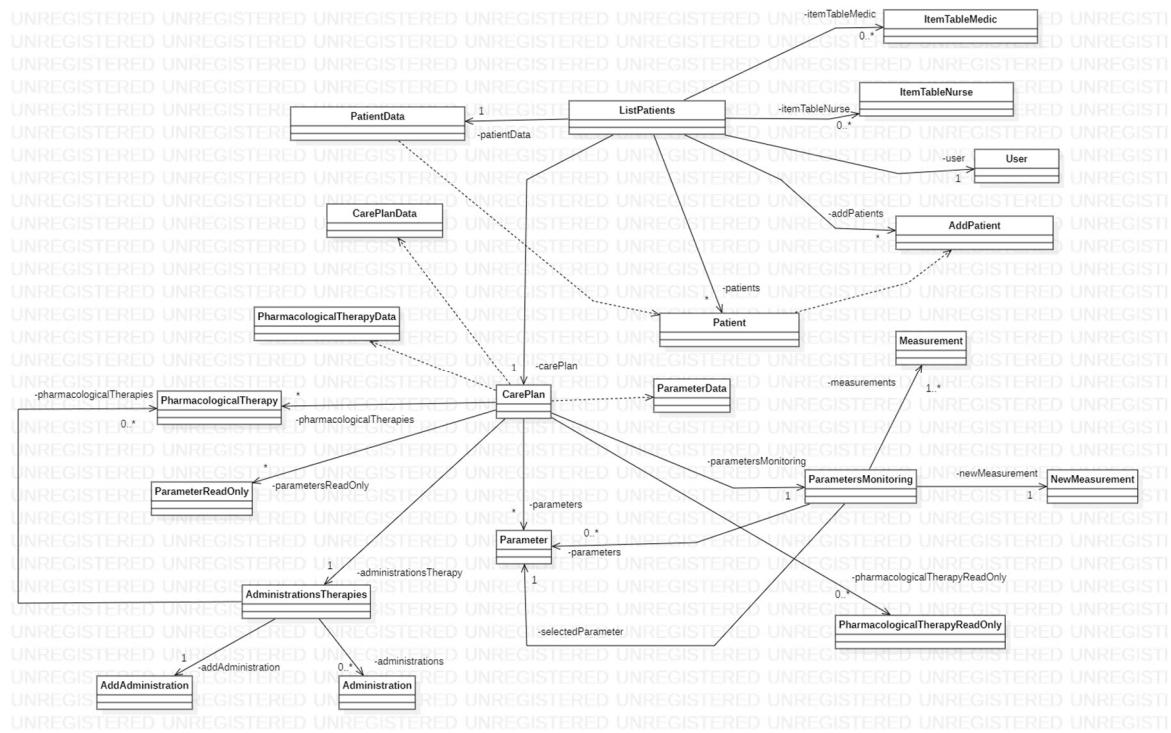
Somministrazione avvenuta con successo

Note

SAVE

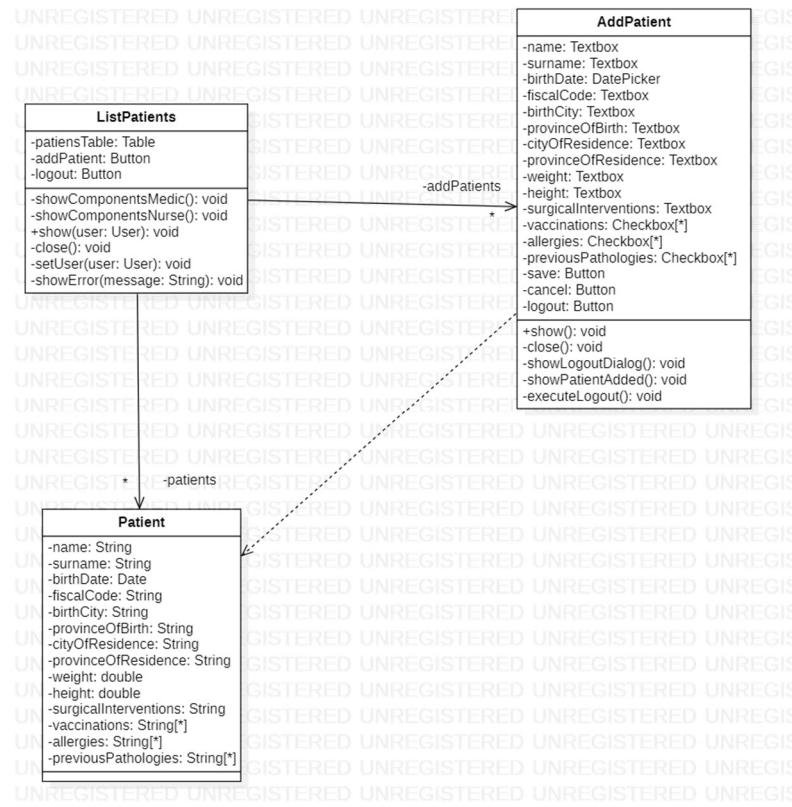
2.6 MODELLO DI DOMINIO

Relazioni tra classi



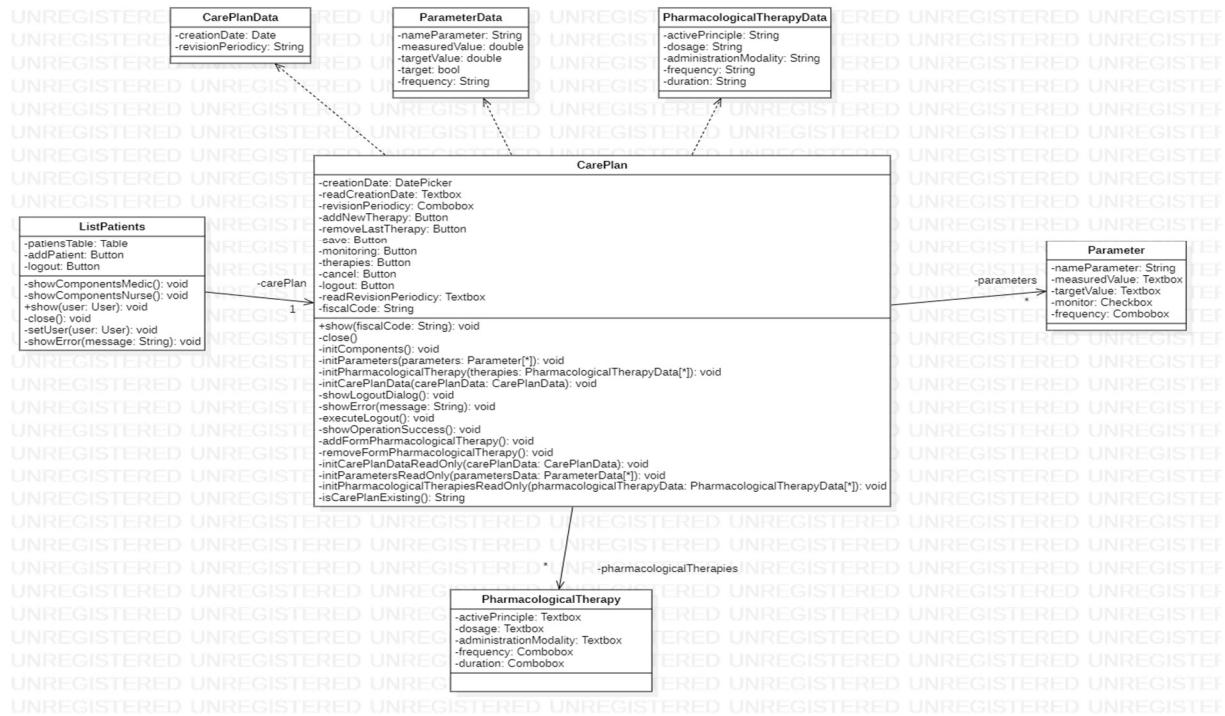
Questo diagramma mostra tutte le classi e relazioni usate nei class diagram successivi.

Aggiungi paziente



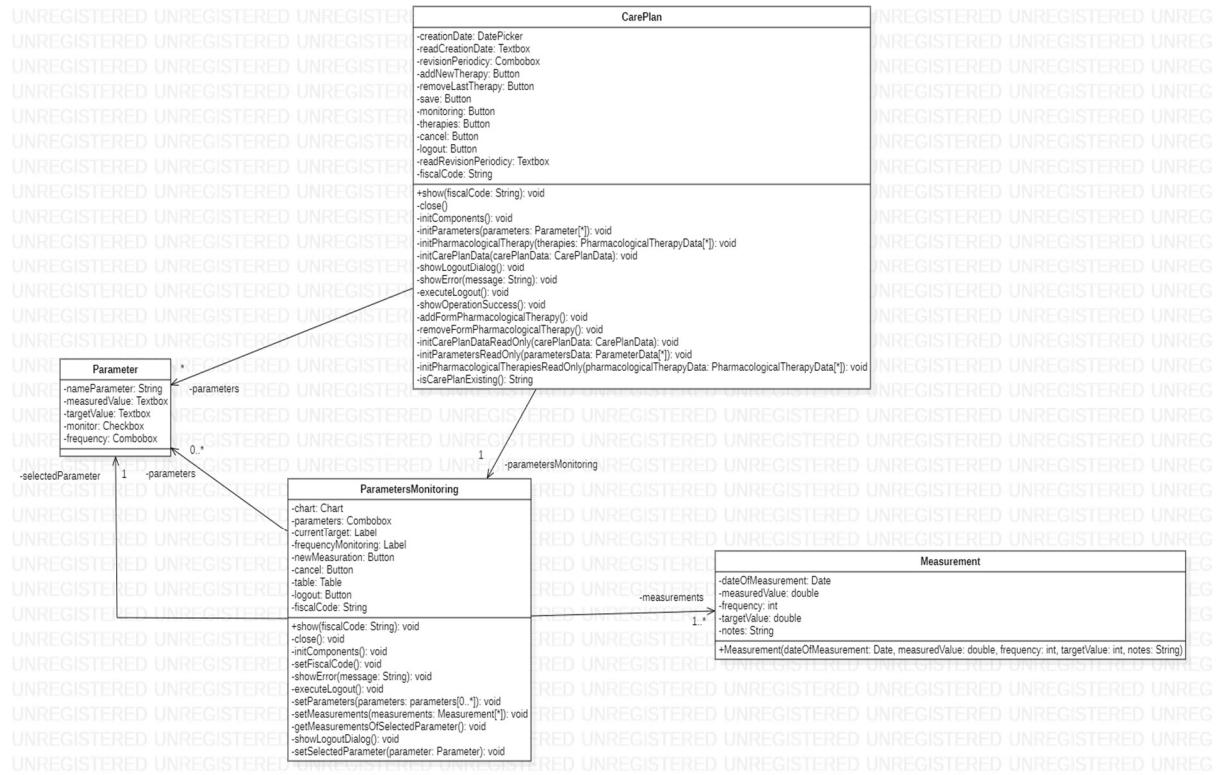
ListPatiens è il componente grafico in cui sono mostrati tutti i pazienti aggiunti precedentemente dal medico, il paziente è rappresentato dell'entità Patient e AddPatient permette al medico di aggiunge un altro paziente al database.

Inserisce-Modifica care plan



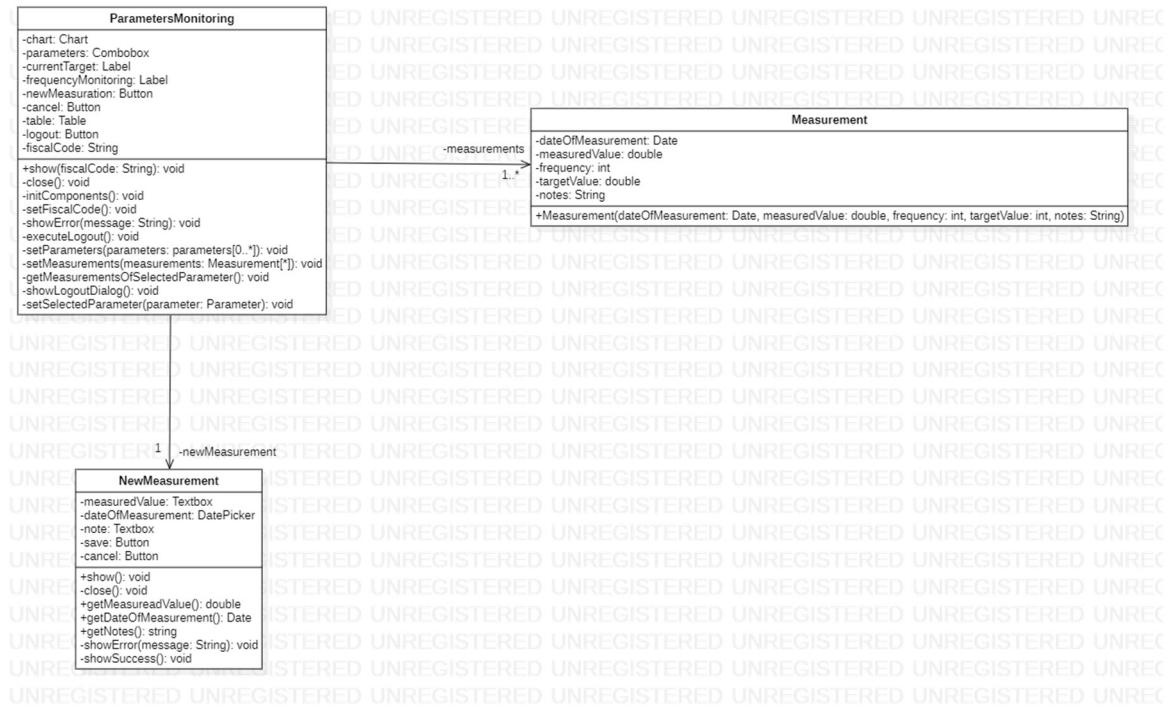
CarePlan è la classe che rappresenta il componente grafico che gestisce la visualizzazione, la modifica e l'inserimento del Care Plan, a seconda del tipo di utente connesso, quindi medico o infermiere, e a seconda dal paziente se ha già associato un Care Plan oppure no. Le classi CarePlanData, ParameterData e PharmacologicalTherapyData sono le entità che si occupano di memorizzare i dati del Care Plan, mentre Parameter e PharmacologicalTherapy rappresentano dei componenti grafici usati dalla classe CarePlan per mostrare i vari dati del Care Plan.

Visualizza misurazioni parametro clinico



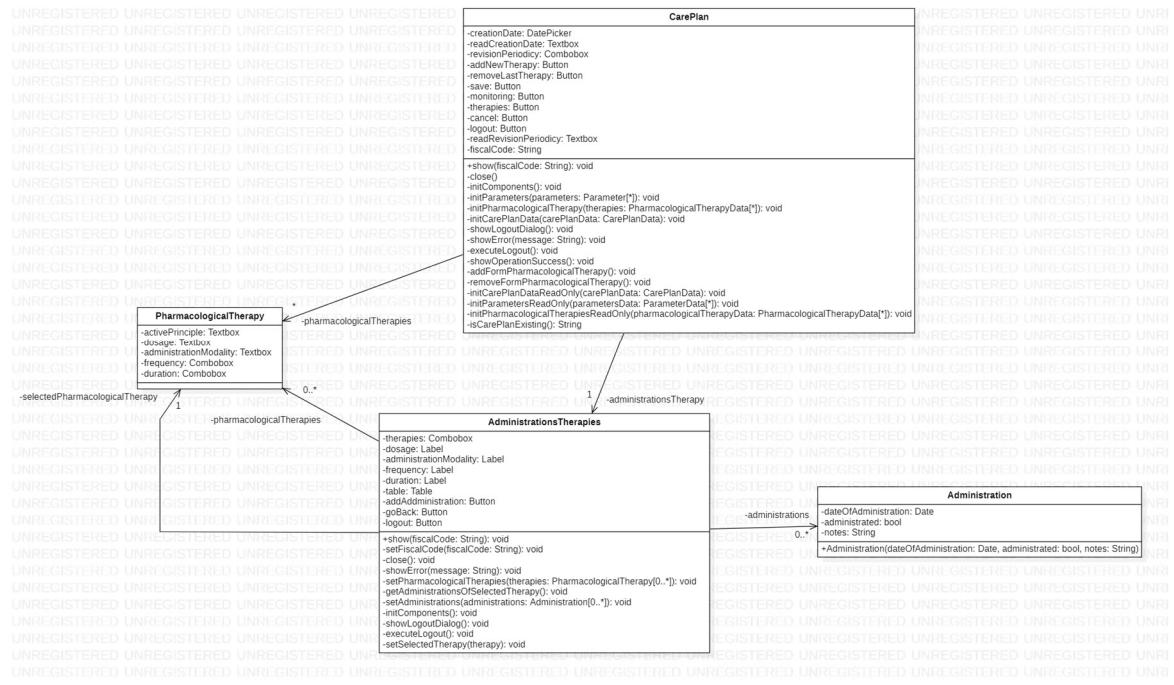
La classe **ParametersMonitoring** è usata per visualizzare il grafico e l'elenco di tutte le misurazioni registrate di un determinato parametro. Per gestire i dati delle misurazioni di un parametro viene usata la classe entità **Measurement**. **ParamatersMonitoring** possiede tutti i parametri etichettati “da monitorare” e anche un campo di tipo **Parameter** che rappresenta il parametro selezionato dall’utente.

Aggiunge misurazione parametro clinico



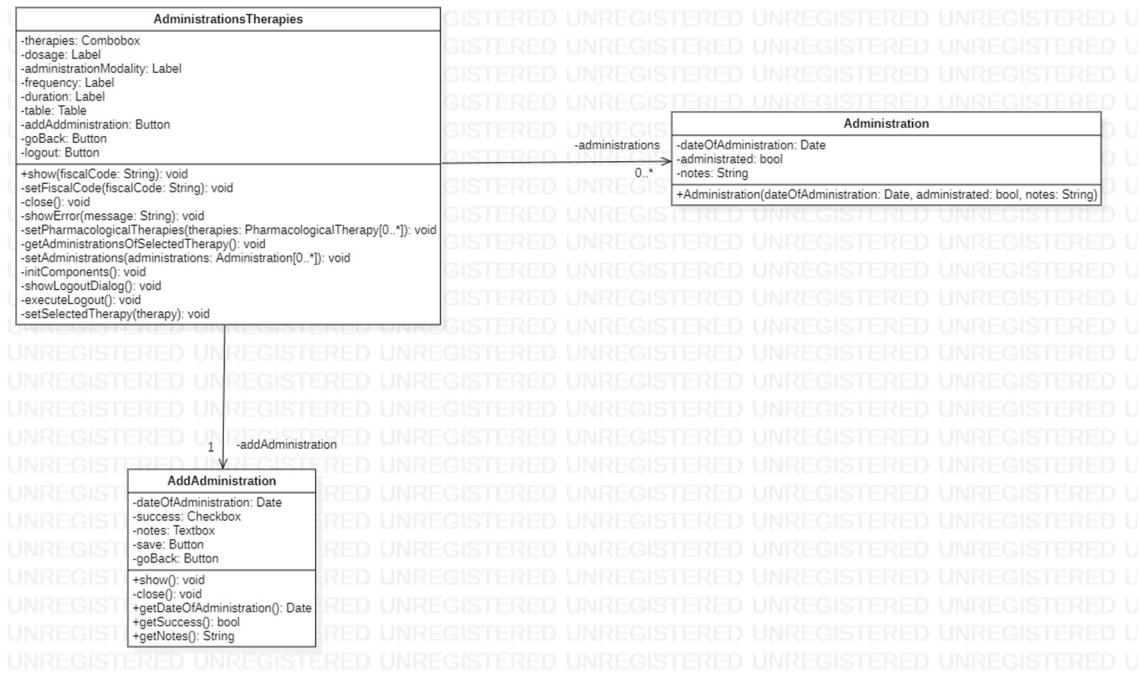
La classe **ParametersMonitoring** per registrare una nuova misurazione fa uso della classe **NewMeasurement**, che rappresenta un componente grafico in cui il medico o l'infermiere inseriscono i valori della nuova misurazione.

Visualizza somministrazioni terapia farmacologica



La classe **AdministrationsTherapies** è usata per visualizzare l'elenco di tutte le somministrazioni registrate di una determinata terapia farmacologica. Per gestire i dati delle somministrazioni di una terapia farmacologica viene usata la classe entità **Administration**. **AdministrationsTherapies** possiede tutte le terapie farmacologiche aggiunte in precedenza dal medico e anche un campo di tipo **PharmacologicalTherapy** che rappresenta la terapia selezionata dall'utente.

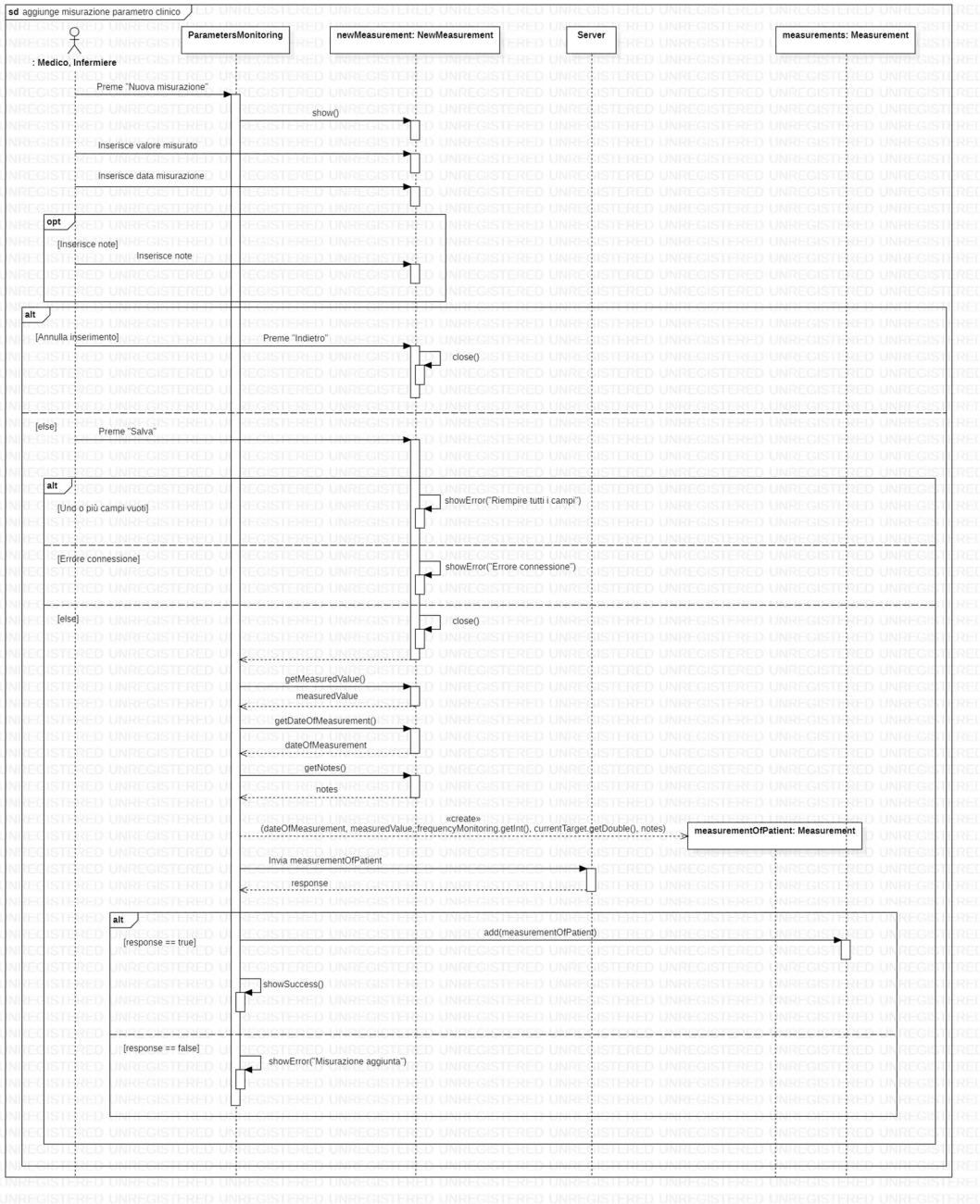
Aggiunge somministrazione terapia farmacologica



La classe AdministationsTherapies per registrare una nuova somministrazione fa uso della classe AddAdministration, che rappresenta un componente grafico in cui l'infermiere inserisce i valori della nuova somministrazione.

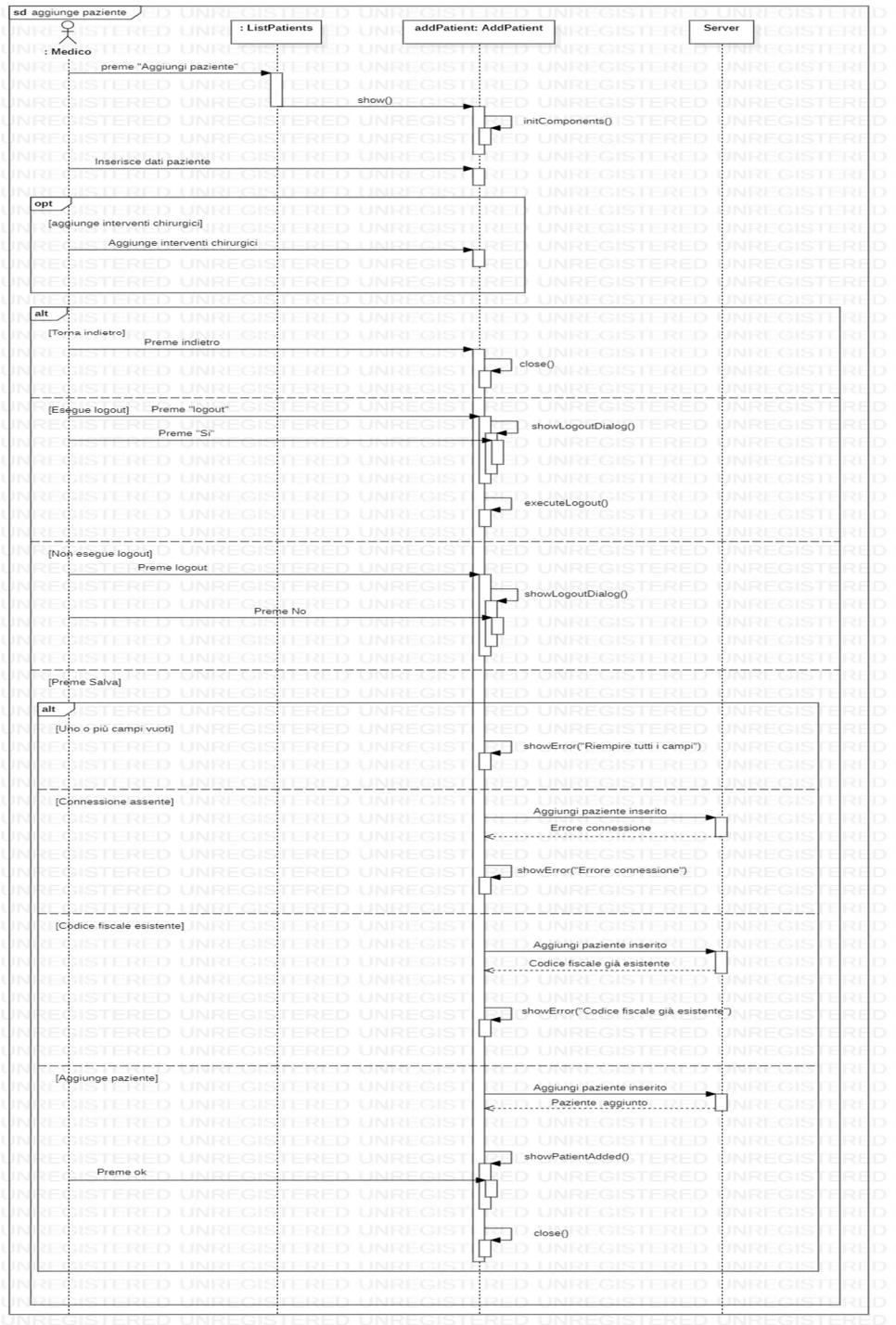
2.7 DIAGRAMMI DI SEQUENZA DI ANALISI

Aggiunge misurazione parametro clinico



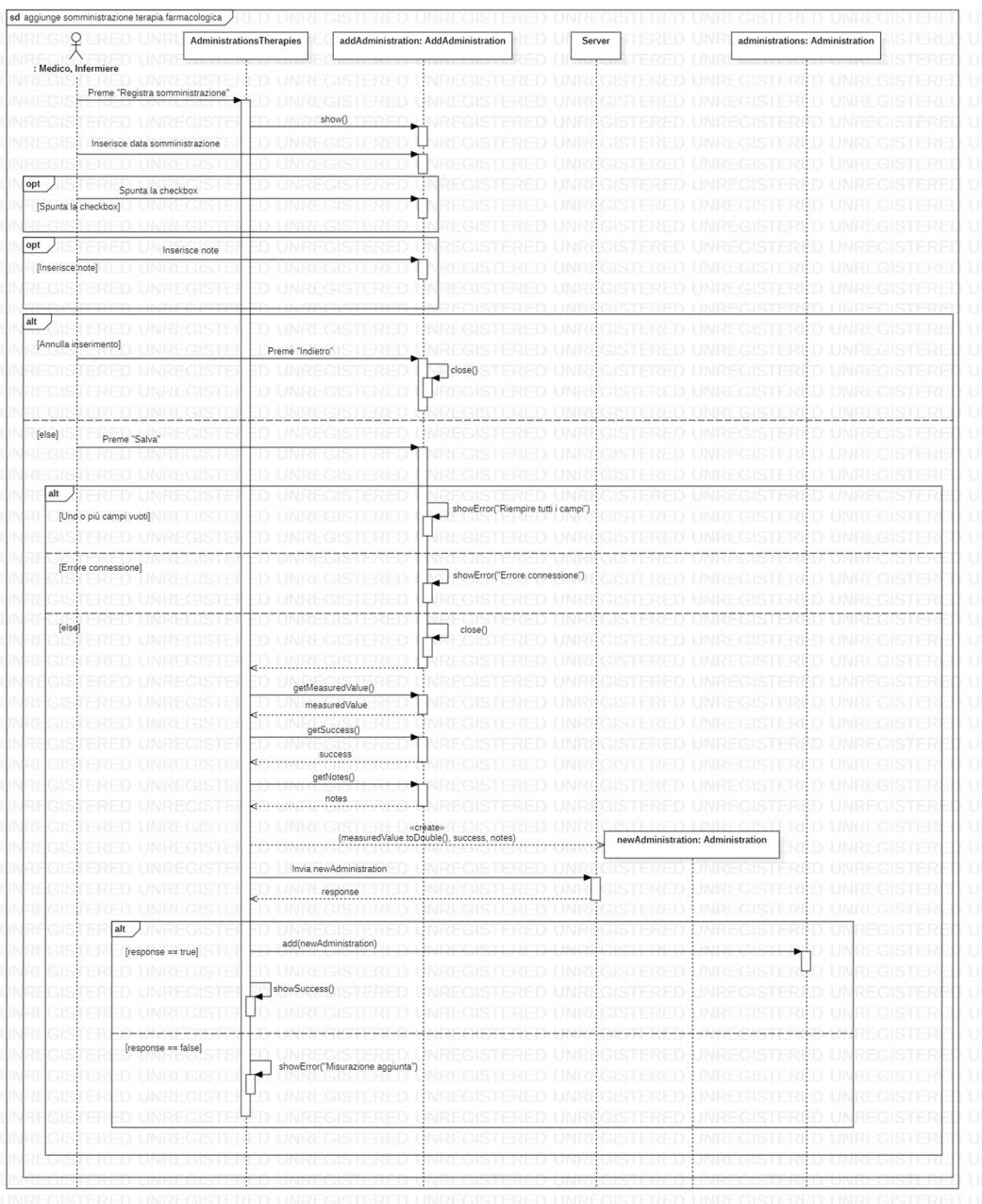
L'utente dopo aver inserito tutti i dati può scegliere se continuare o no la registrazione della misurazione. Se continua verranno effettuati dei controlli per verificare la correttezza dei dati inseriti e successivamente la classe ParametersMonitoring ottiene i dati da newMeasurement, crea l'oggetto measurementOfPatient e lo aggiunge. Se l'operazione va a buon fine, la misurazione verrà aggiunta e si visualizzerà un messaggio di successo, altrimenti un messaggio di errore.

Aggiunge paziente



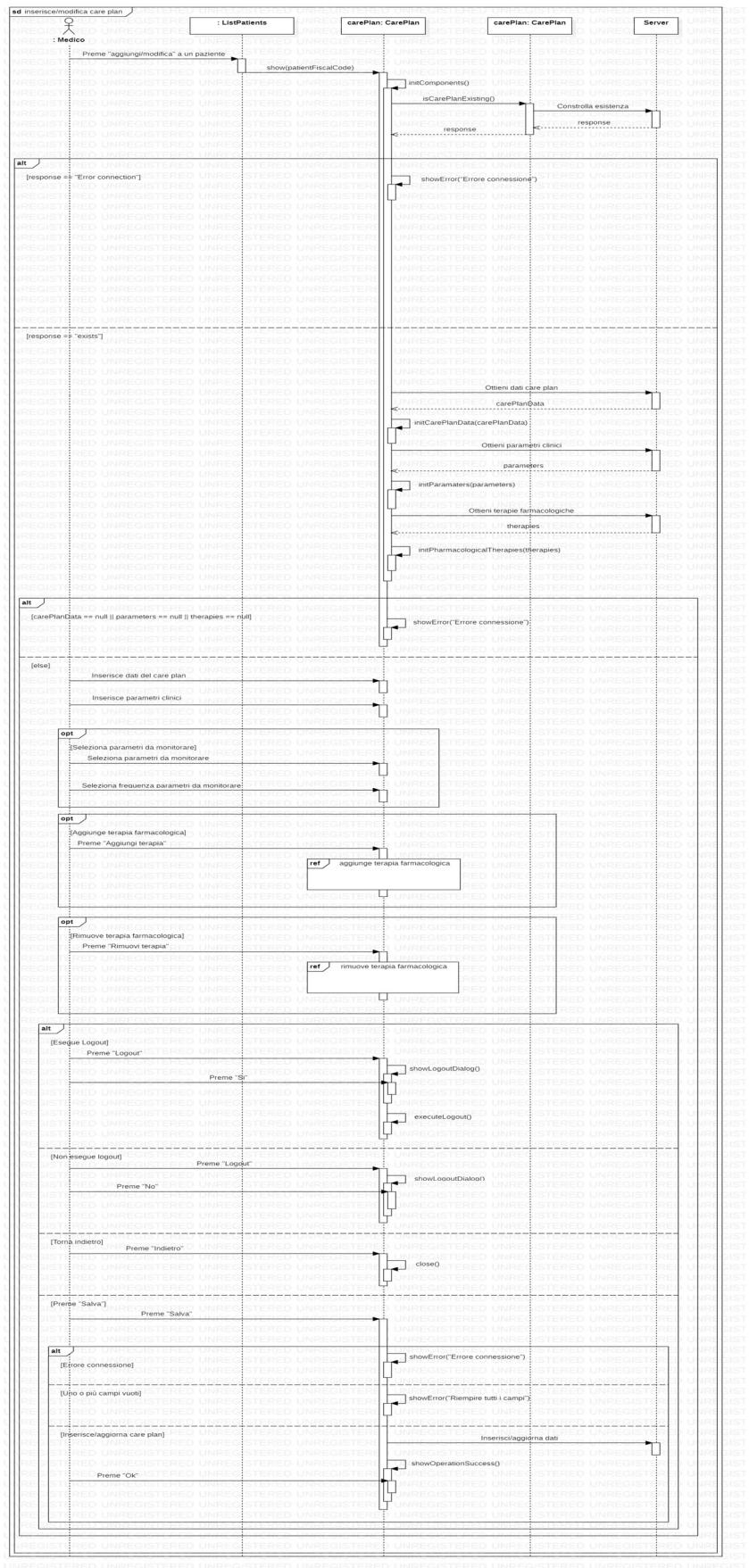
Il medico tramite il componente grafico AddPatient, può inserire i dati del nuovo paziente. Dalla stessa schermata può anche annullare l'operazione, oppure effettuare il login. Nel caso in cui il medico decida di confermare l'aggiunta del nuovo paziente verranno effettuati dei controlli di verifica dei dati immessi, se la verifica va a buon fine e se non si hanno errori di connessione, il paziente verrà aggiunto e il medico vedrà un messaggio di successo, altrimenti verrà mostrato un messaggio di errore.

Aggiunge somministrazione terapia farmacologica



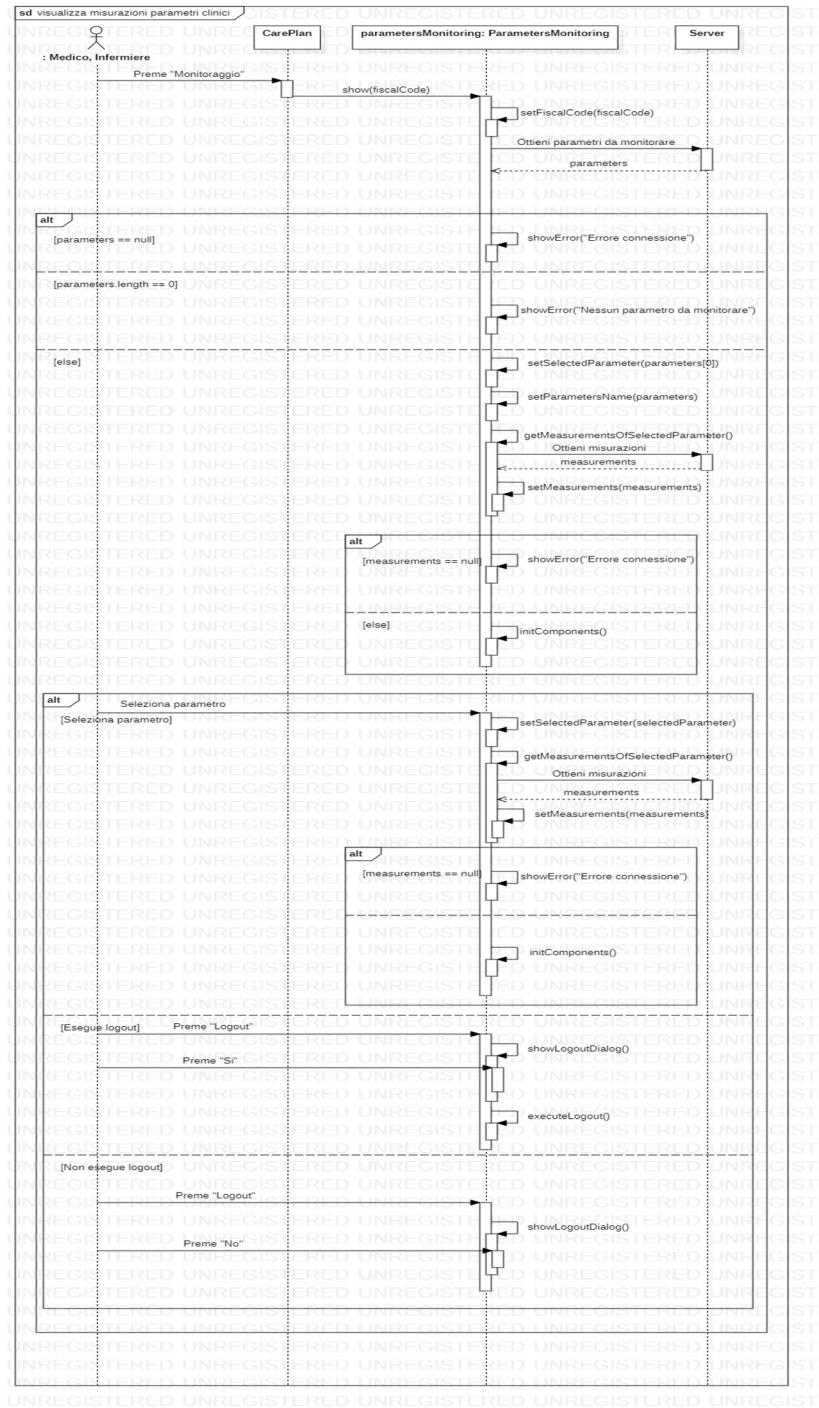
L'utente dopo aver inserito tutti i dati può scegliere se continuare o no la registrazione della somministrazione. Se continua verranno effettuati dei controlli per verificare la correttezza dei dati inseriti e successivamente la classe AdministrationTherapies ottiene i dati da addAdministration, crea l'oggetto newAdministration e lo aggiunge. Se l'operazione va a buon fine, la misurazione verrà aggiunta e si visualizzerà un messaggio di successo, altrimenti un messaggio di errore.

Inserisce-modifica care plan



Come prima cosa, CarePlan verifica se il paziente ha già associato un proprio Care Plan, un caso positivo, vengono ottenuti i dati del Care Plan e si verifica che non ci siano errori di connessione. Da qui il medico può decidere se proseguire con l'inserimento/modificare dei vari dati obbligatori e opzionali del Care Plan, oppure può annullare l'operazione o può eseguire il logout. Nel caso si decida di continuare, si procede alla verifica della correttezza dei campi inseriti, nel caso in cui non ci sono errori, il Care Plan viene inserito/modificato e si mostra un messaggio di successo, altrimenti si mostra un messaggio di errore

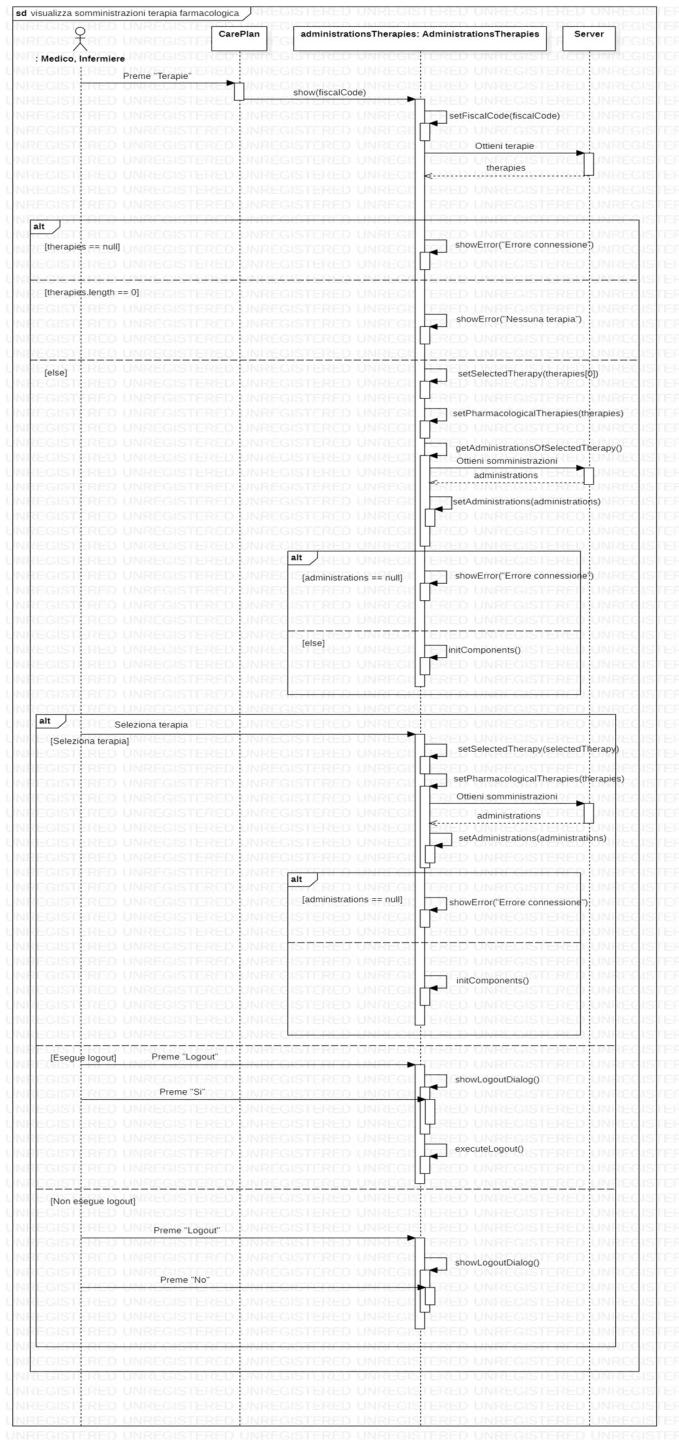
Visualizza misurazioni parametri clinici



L'utente tramite il componente ParametersMonitoring può selezionare uno dei parametri etichettati come "da monitorare", selezionato un parametro il componente si occuperà di mostrare tutte le

misurazioni effettuate in forma grafica e tabellare. Se ci sono errori di connessione sarà mostrato un messaggio di errore.

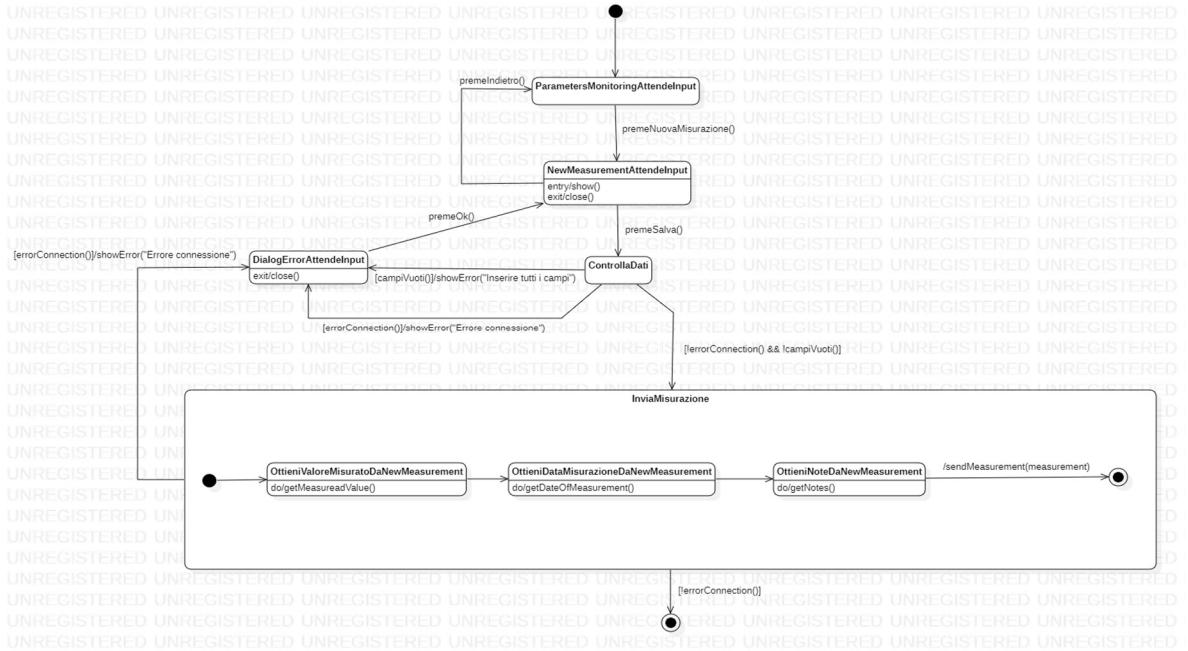
Visualizza somministrazioni terapia farmacologica



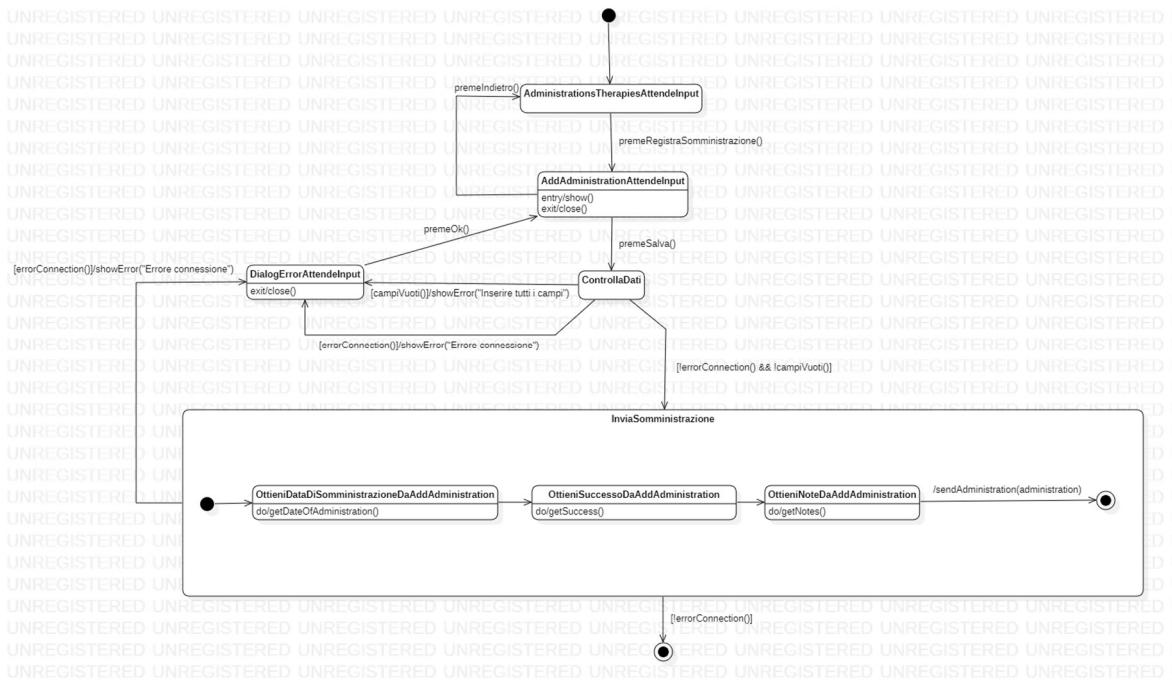
L'utente tramite il componente Administration può selezionare uno dei parametri etichettati come "da monitorare", selezionato un parametro il componente si occuperà di mostrare tutte le misurazioni effettuate in forma grafica e tabellare. Se ci sono errori di connessione sarà mostrato un messaggio di errore.

2.8 DIAGRAMMI DI STATO DI ANALISI

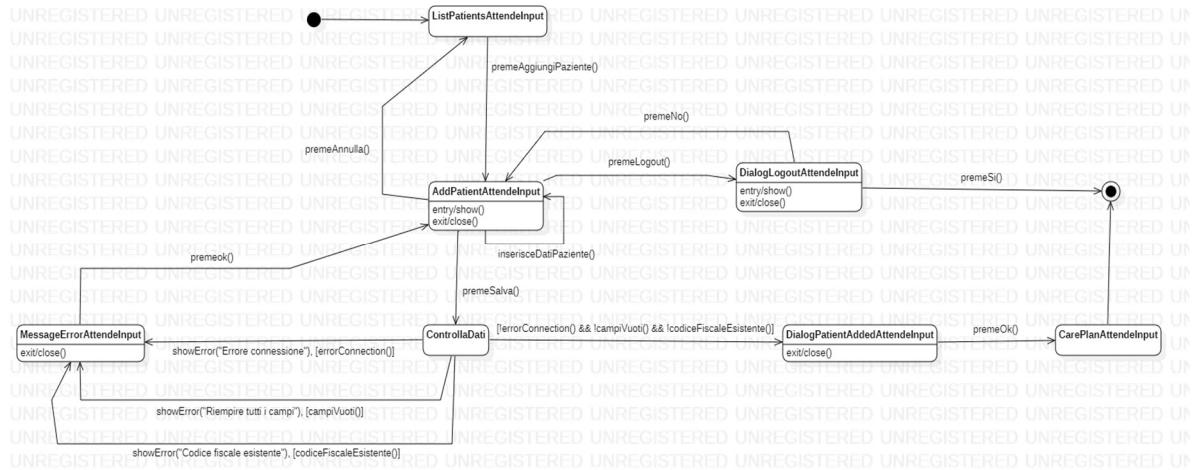
Aggiunge misurazione parametro clinico



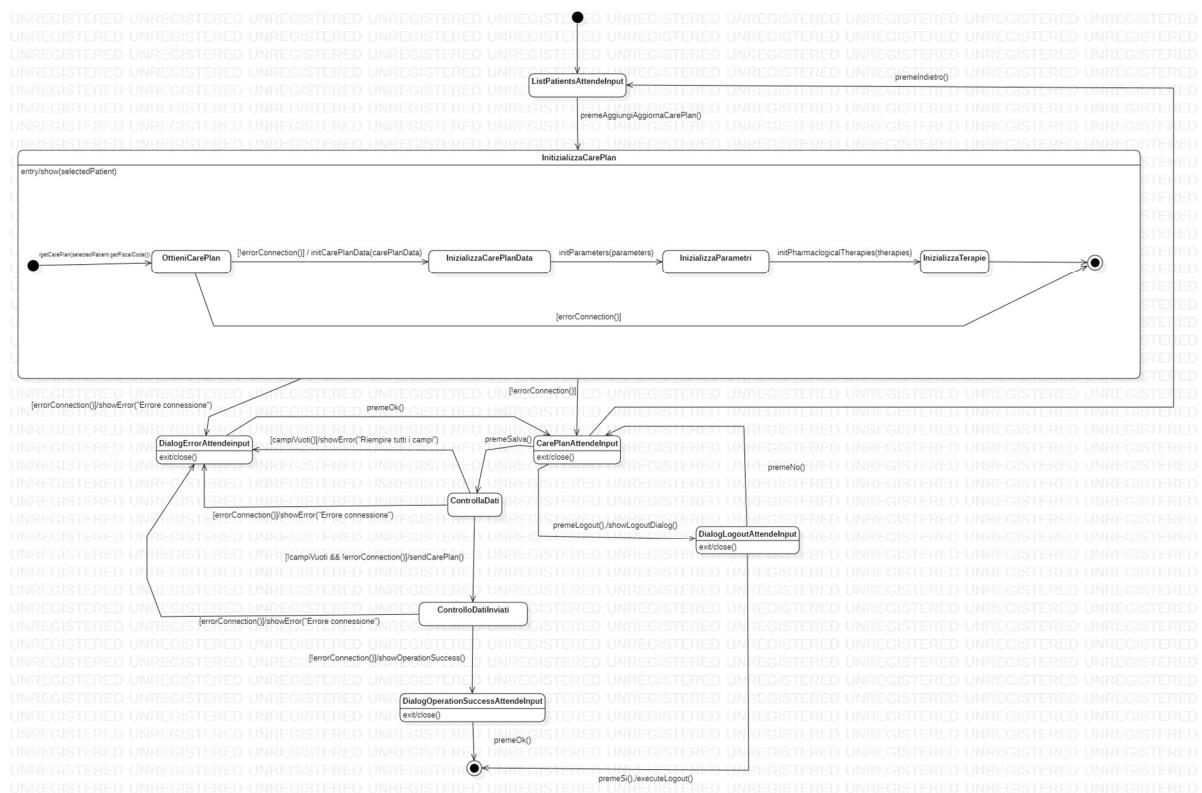
Aggiunge somministrazione terapia farmacologica



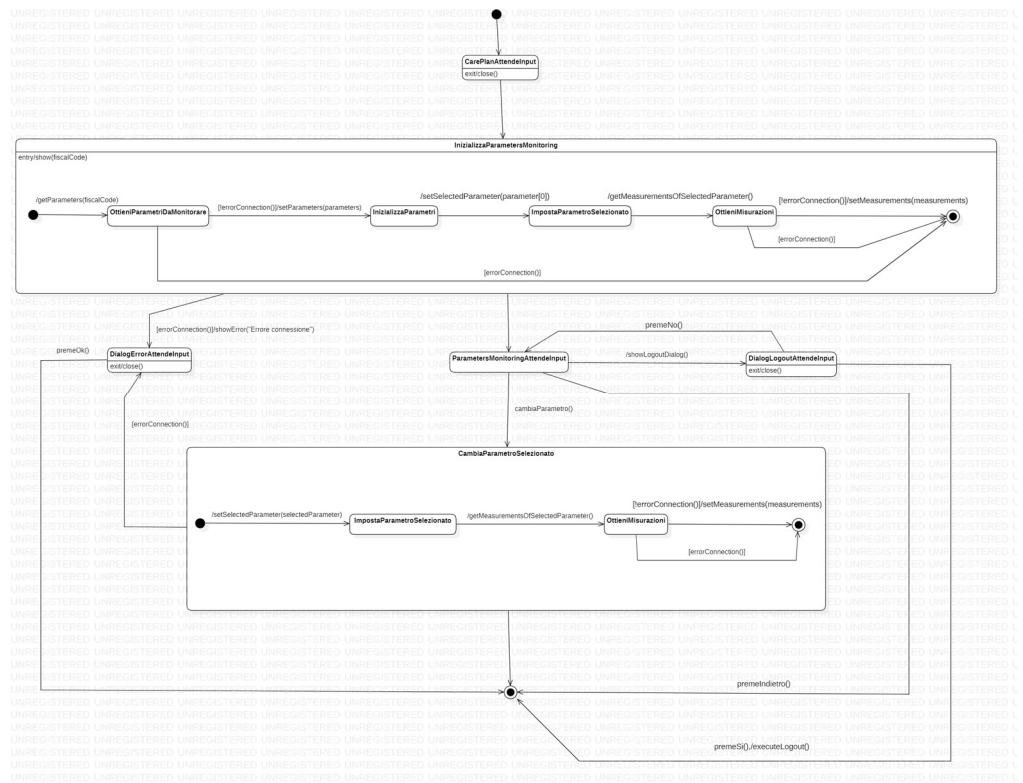
Aggiungi paziente



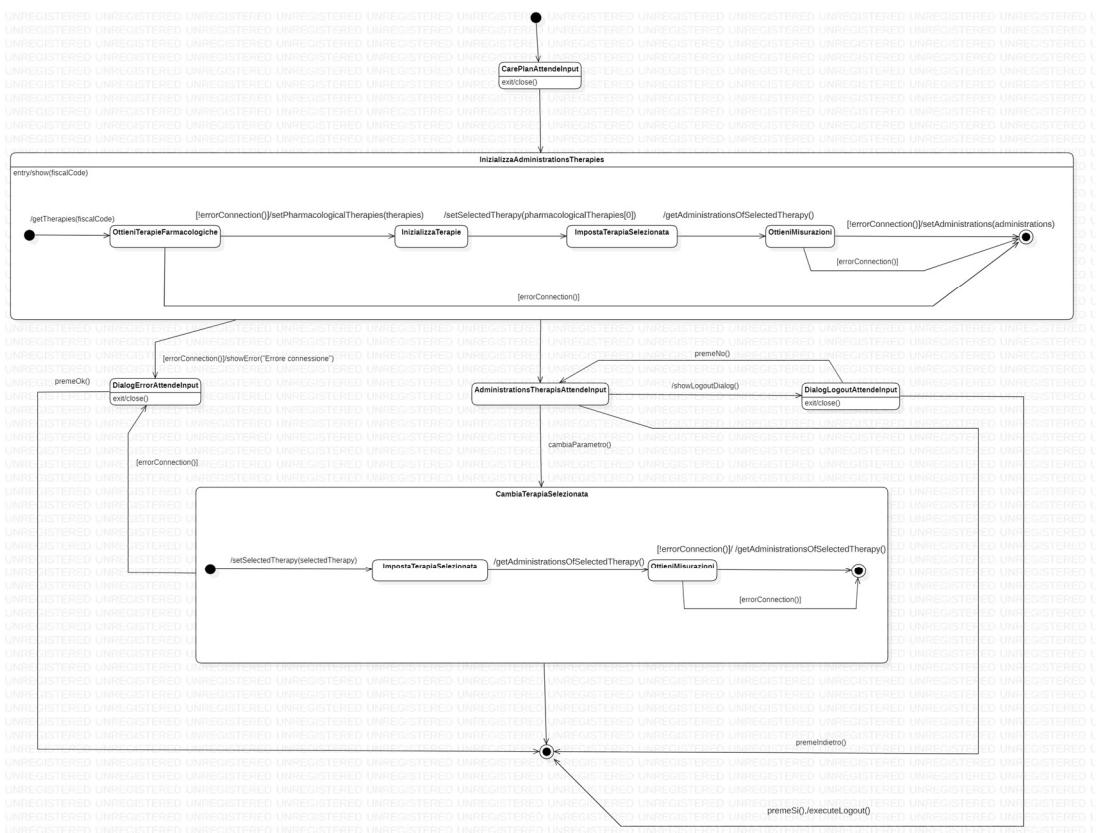
Inserisce-modifica care plan



Visualizza misurazioni parametri clinici

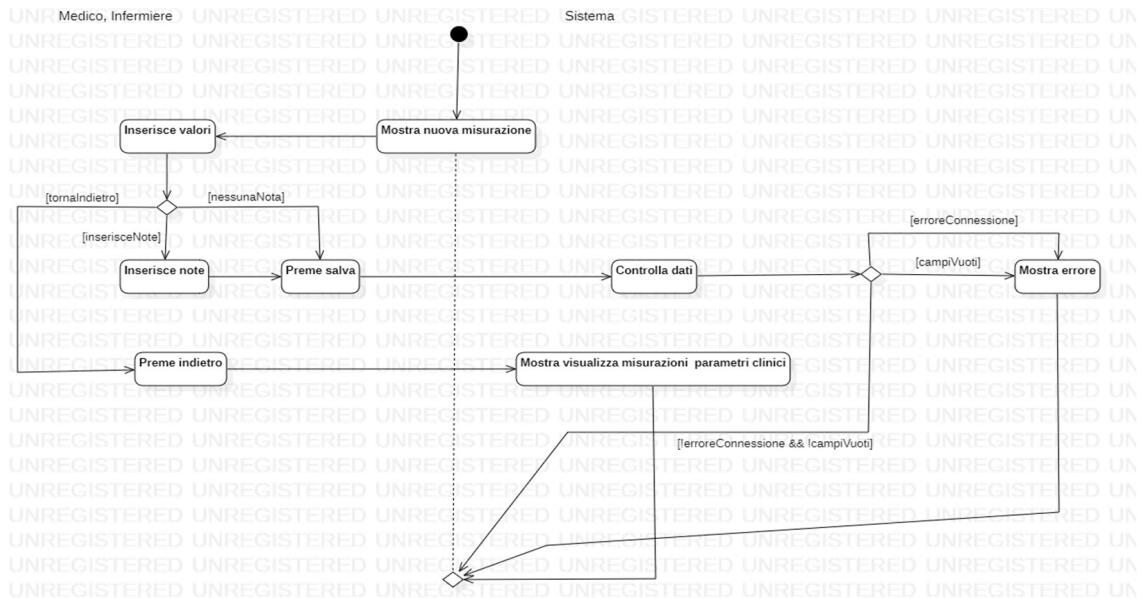


Visualizza somministrazioni terapia farmacologica

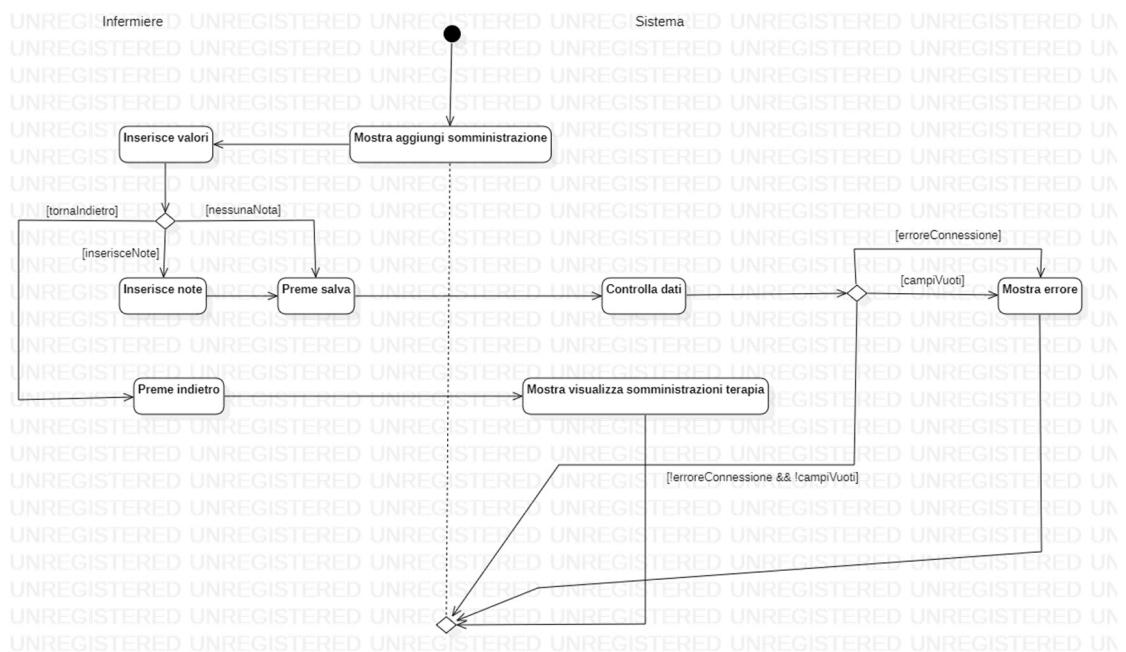


2.9 DIAGRAMMI DI ATTIVITÀ

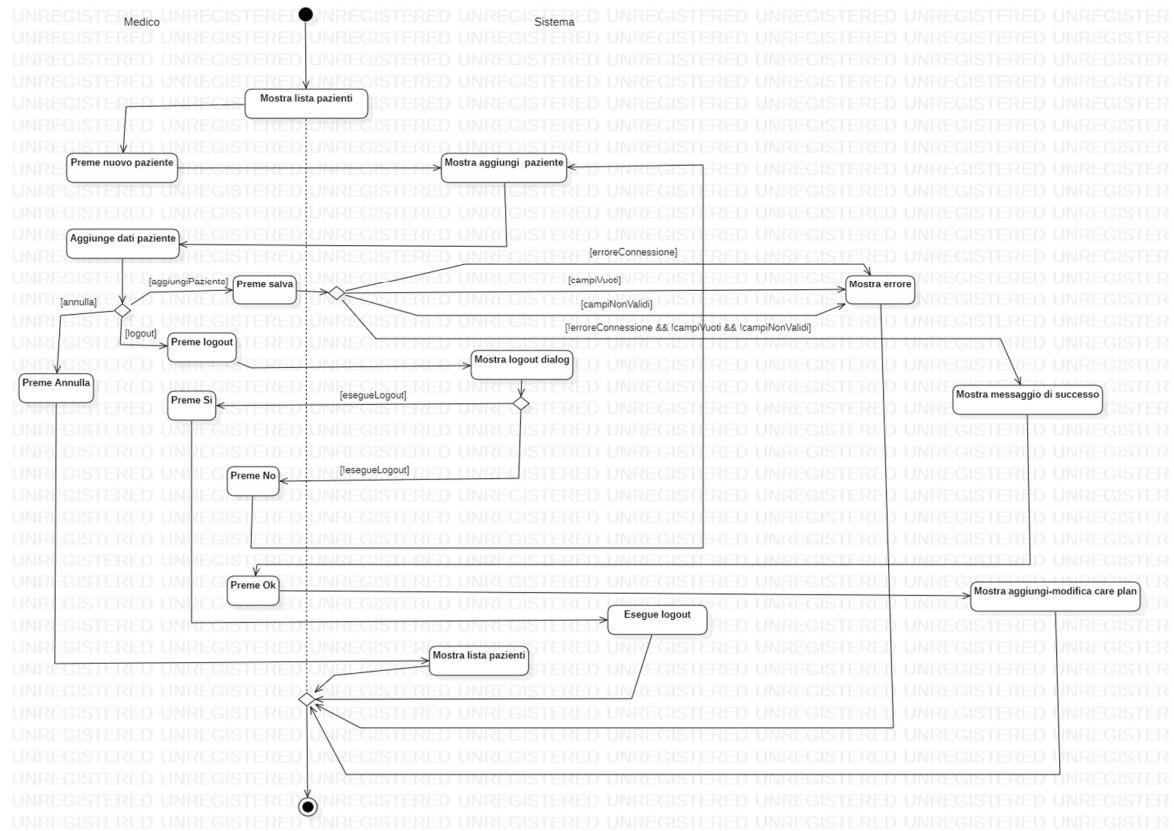
Aggiunge misurazione parametro clinico



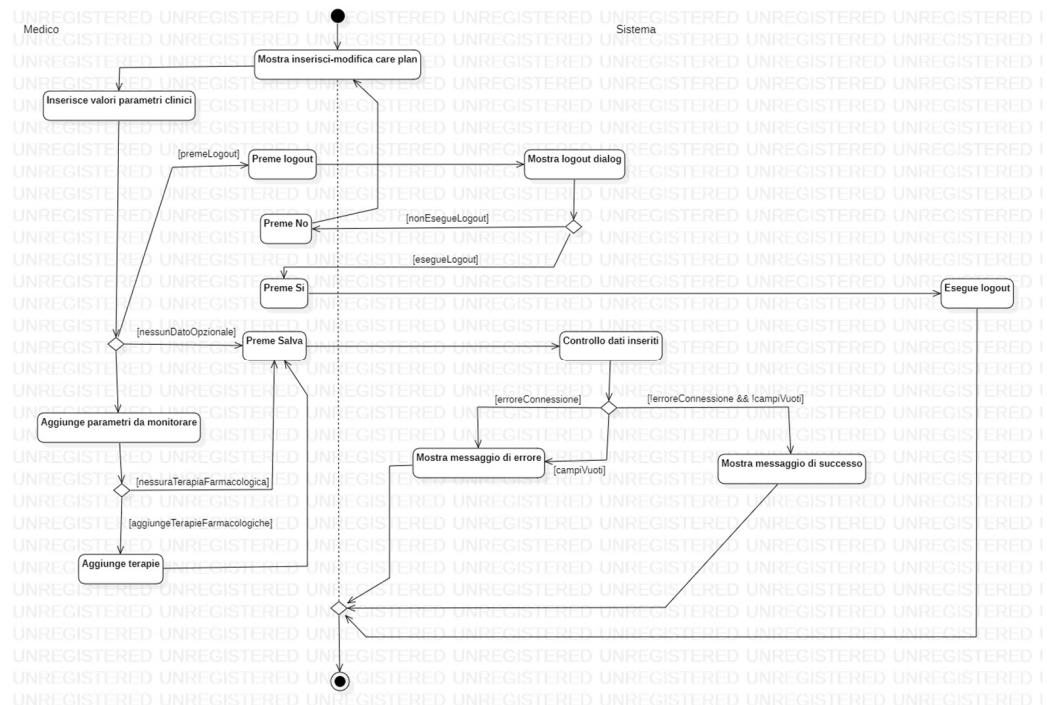
Aggiunge somministrazione terapia famacologica



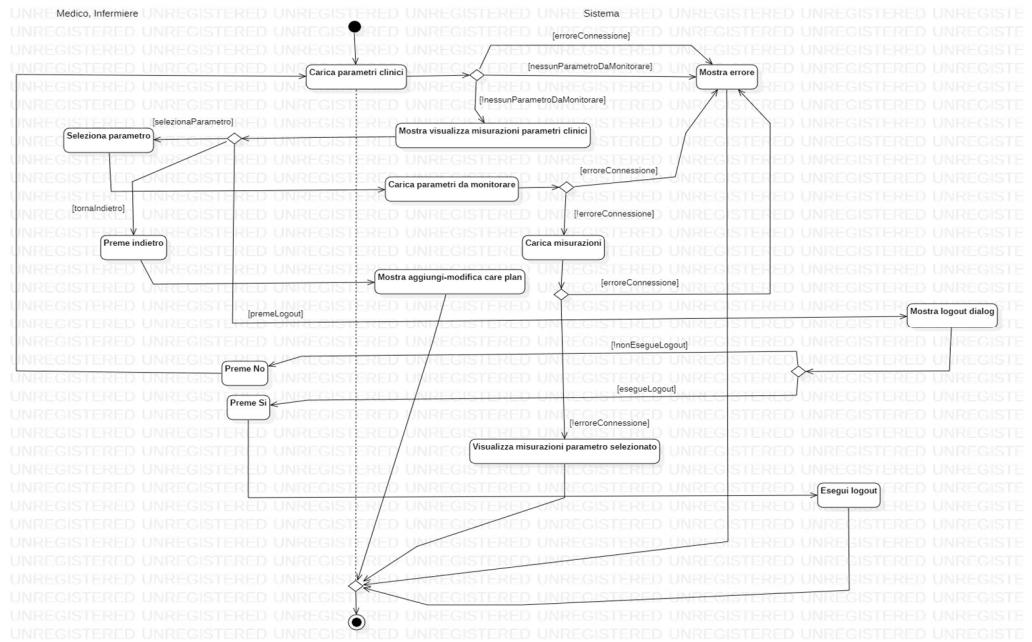
Aggiungi paziente



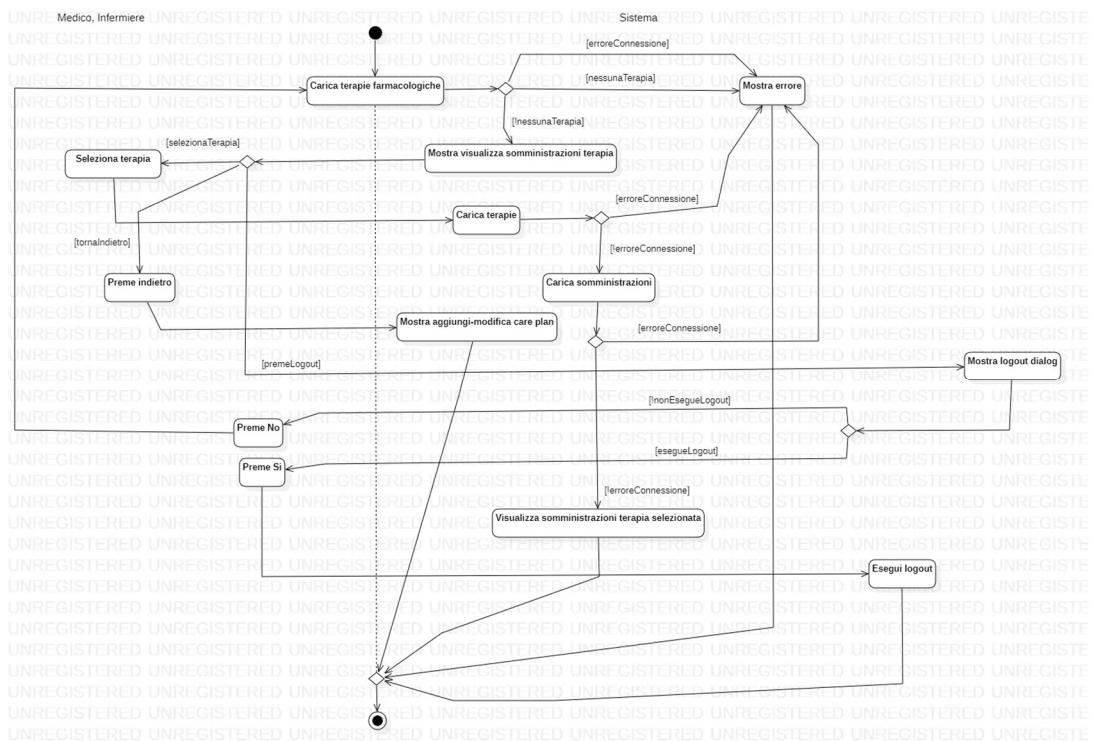
Inserisce-modifica care plan



Visualizza misurazioni parametri clinici



Visualizza somministrazioni terapia farmacologica



CAPITOLO: 3 IMPLEMENTAZIONE E TECNOLOGIE

3.1 ARCHITETTURA CLIENT-SERVER

L'architettura software utilizzata è quella client-server. Essa è un tipo di architettura che permette di richiedere, o di ricevere, un determinato servizio da un server centralizzato. I dispositivi client forniscono un'interfaccia all'utente di inviare richieste al server, e successivamente permette di visualizzare il risultato del server di una precedente richiesta del client. I vantaggi di questo tipo di architettura sono:

- Centralizzazione, tutti le necessarie informazioni sono piazzate in un singolo posto, questo permette di avere il pieno controllo dei dati da gestire e da amministrare
- Sicurezza, i dati sono ben protetti grazie all'architettura centralizzata. Essa può essere rafforzata con dei controlli di accesso
- Scalabilità, è possibile aggiungere nuove risorse senza l'interruzione del sistema
- Gestione, siccome tutti i dati sono gestiti da un server centrale, è davvero semplice tracciarli e gestirli
- Accessibilità, indipendentemente dalla piattaforma o dal luogo, ogni client può accedere al sistema

3.2 COMUNICAZIONE HTTP

I messaggi http sono come i dati vengono scambiati tra client e server. Ci sono due tipi di messaggi:

- Richieste, esse sono inviate dal client per far sì che riceva dal server un responso
- Responsi, le risposte del server

I messaggi http sono composti da informazioni testuali codificate in ASCII e si estendono su multiple linee. Gli sviluppatori web raramente costruiscono questi messaggi da soli, in generale software come web browser eseguono questa azione.

Le richieste e responsi http condividono una struttura similare e sono composti di:

- Una linea iniziale, chiamata header, che descrive le richieste da essere implementate, o il suo stato che può essere di successo o fallimento;

- Un insieme opzionale di intestazioni http che specificano la richiesta, o descrivono il corpo, incluso il messaggio;
- Una linea bianca che indica che tutte le meta-information sono state informate;
- Un corpo opzionale, chiamato body, che contiene dati associati con la richiesta, o il documento associato con un response. La presenza del corpo e la sua grandezza è specificata nell'header.

3.3 JSON

JSON (JavaScript Object Notation) è un formato utilizzato per immagazzinare e inviare dati al server o al client. Esso utilizza un formato leggibile per l'essere umano che consiste in una collezione di coppie chiave-valore oppure da array. Il formato JSON è sintatticamente identico al codice per creare oggetti in Javascript.

3.4 REST API

Una REST API è un application programming interface che conforma al vincolo dello stile architetturale REST e permette l'interazione con servizi web RESTful. REST sta per representational state transfer. REST è un insieme di vincoli architetturali, esso non è un protocollo o uno standard. Quando una richiesta del client è effettuata attraverso una API RESTful, essa trasferisce una rappresentazione dello stato della risorsa al richiedente o all'endpoint. Questa informazione può essere inviata tramite il formato JSON attraverso messaggi http. L'intestazione e i parametri sono importanti nei metodi http di una richiesta RESTful API, per esempio possono contenere informazioni come: metadata, autorizzazione, URI, cookie, caching e altro. Per quanto riguarda un response header. Ci sono request header e response header, ognuno dei quali con le loro informazioni di connessione http e status code. Una API è considerata RESTful se:

- client e server comunicano tramite messaggi http;
- la comunicazione tra client e server è stateless;
- i dati possono essere memorizzati nella cache semplificando l'interazione client-server;
- un'interfaccia uniforme tra componenti così che le informazioni sono trasferite in una forma standard;

- un sistema stratificati che organizza ogni tipo di server coinvolto nel recupero di informazioni richieste in gerarchie, invisibili ai client.

3.5 SICUREZZA E AUTENTICAZIONE

3.5.1 HASHING PASSWORD

Quando l'utente esegue la creazione di un nuovo account, la password non viene memorizzata in chiaro, ma prima di essere memorizzata vengono eseguite delle operazioni di criptaggio. Per fare ciò, è stata utilizzata una funzione che permette l'hashing della password chiamata bcrypt. Essa è basata sul cifrario Blowfish, oltretutto esso incorpora una chiave di sale per garantire la protezione contro attacchi rainbow table. Una volta effettuato l'hashing la password avrà un formato del genere:

```
$2b$[cost]$[22 character salt][31 character hash]
```

Per esempio:

```
$2a$10$N9qo8uLoickgx2ZMRZoMyeIjZAgcf17p921dGxad68LJZdL17lhWy
```

Dove:

- \$2a\$: è l'algoritmo hash identificatore;
- 10: fattore costo;
- N9qo8uLoickgx2ZMRZoMye: chiave di sale;
- IjZAgcf17p921dGxad68LJZdL17lhWy: stringa a cui applicata l'hash.

3.5.2 FUNZIONAMENTO HASHING

Quando l'utente effettua la richiesta della creazione dell'account, il server, dopo aver ricevuto i dati dal client e dopo aver effettuato i dovuti controlli, utilizza la funzione `bcrypt.genSalt(costo)` per generare la chiave di sale, successivamente tramite la funzione `bcrypt.hash(password, chiaveDiSale)` genera la password criptata e quest'ultima sarà memorizzata nel database.

Quando l'utente effettua la richiesta di accesso, il server tramite la funzione `bcrypt.compare(password, passwordCriptata)` permette di verificare se la password immessa dall'utente corrisponde a quella memorizzata nel database, essa restituisce un valore booleano.

3.5.3 AUTENTICAZIONE UTENTE

Uno degli aspetti più critici di una web application, riguardo alla sicurezza, è quello di verificare che un client, che ha effettuato una richiesta al server, abbia i permessi di usufruire di quel determinato servizio. Per verificare ciò è stata creata una tabella nel database associata all'utente chiamata Sessions. Essa possiede come colonne i campi che permettono la memorizzazione di un token di accesso, della data di creazione del token e la data di scadenza del token. Il token di accesso verrà successivamente usato per controllare se l'utente può accedere o no a un determinato servizio offerto dal server. Ogni token ha anche una data di scadenza, essa reclama che ogni richiesta che viene effettuata dopo questa data deve essere non accettata e quindi non processata. Per la creazione dei token è stato usato JSON Web Token (JWT). Esso è uno standard che definisce un modo compatto e un modo auto-contenuto per la trasmissione sicura di informazioni tra due parti come oggetti JSON. Questa informazione può essere verificata perché è firmata digitalmente.

3.5.4 FUNZIONAMENTO AUTENTICAZIONE

Quando l'utente effettua l'accesso alla web application, il server ottiene dal database la sessione dell'utente, con il token di accesso e la data di creazione e di scadenza. Successivamente, il server, controlla se il token è scaduto, se esso non lo è lo invia al client, altrimenti ne viene generato un altro tramite la funzione

`jwt.sign(payload, chiavePrivata)`. Il payload è un oggetto o una stringa che verrà firmato e la chiavePrivata sarà utilizzata per codificare o decodificare il token. La chiave è stata memorizzata in un file .env, quando il server verrà avviato essa sarà caricata in memoria come variabile di ambiente. Dopo ciò verranno create la data di creazione e la data di scadenza, quest'ultima sarà uguale alla data corrente + 2 giorni. Il tutto sarà memorizzato nel database e il token sarà mandato al client. Il client memorizzerà il token localmente nel browser. Quando il client effettuerà una richiesta, mandando il proprio token al server, quest'ultimo verificherà tramite la funzione `jwt.verify(token, chiavePrivata)` se il token è valido oppure no, se il token è valido essa restituisce il payload decodificato.

3.5.5 MIDDLEWARE

Il framework Express permette di definire un middleware, applicabile a una rotta del server. Un middleware è una funzione che esegue delle istruzioni prima che il server esegua la richiesta del server. In questo caso per autenticare la richiesta di un client, è stato applicato un middleware che verificasse l'autenticità del token del client. Grazie ai middleware, è possibile definire delle rotte private, cioè che possono essere da un client se e solo se hanno i livelli di autorizzazione necessari.

3.5.6 LOGOUT

Quando un client invia al server una richiesta di logout, quest'ultimo, come prima cosa, verificherà tramite middleware se il token è corretto, se esso è corretto il server rimuoverà la sessione corrente dell'utente dal database, in seguito il client, rimuoverà il token memorizzato localmente nel browser.

3.6 PARADIGMI DI PROGRAMMAZIONE

3.6.1 LATO SERVER

Per sviluppare il server di questa web application è stato utilizzato il linguaggio Typescript. Typescript è un linguaggio di programmazione che permette lo sviluppo tramite la programmazione orientata agli oggetti. Esso è il paradigma di programmazione utilizzato per sviluppare il server. La programmazione orientata agli oggetti è basata sul concetto di oggetti, i quali contengono dati e codice, i dati in forma di campi e il codice in forma di metodi.

3.6.2 LATO CLIENT

Per sviluppare il client è stato utilizzato il linguaggio Javascript. Javascript è un linguaggio che supporta la OOP, ma non è esattamente come quella classica implementata da altri linguaggi. Allora il paradigma utilizzato per il client è quello di tipo procedurale. Esso è un paradigma derivato dal paradigma imperativo, basato sul concetto di chiamate di procedure. Una procedura contiene una serie di passi computazionali da essere eseguiti. Ogni procedura può essere chiamata a ogni punto durante l'esecuzione di un programma.

3.7 DOM E VIRTUAL DOM

3.7.1 DOM

Il Document Object Model (DOM) è un API usato nella programmazione per documenti HTML e XML. Esso definisce la struttura logica di documenti e il modo in cui un documento è acceduto e manipolato. Con il Document Object Model, gli sviluppatori software possono creare e costruire documenti, navigare nella loro struttura, aggiungere, modificare ed eliminare elementi e contenuto. Qualsiasi cosa trovato in un documento HTML o XML, esso può essere acceduto, cambiato, cancellato o aggiunto usando il Document Object Model, con qualche eccezione. In particolare, le interfacce DOM per il subset interno ed esterno ancora non sono state specificate. Il DOM rappresenta un documento con un albero logico. Ogni ramo dell'albero termina con un nodo e ogni nodo contiene un oggetto. I metodi DOM permettono di accedere all'albero in un modo programmatico. I nodi

possono avere dei gestori degli eventi attaccati a loro. Una volta che l'evento accade, il gestore dell'evento viene eseguito.

3.7.2 VIRTUAL DOM

Il Virtual DOM (VDOM) è un concetto di programmazione dove una virtuale rappresentazione dell'UI che è tenuta in memoria è sincronizzata con il DOM reale da una libreria, come per esempio ReactDOM. Quando accade un cambiamento in un componente React, esso non aggiorna il DOM reale, invece React crea e tiene una copia del DOM reale in memoria, il Virtual DOM. Dopo dei cambiamenti, esso confronta il vecchio DOM con il nuovo Virtual DOM e aggiorna solo le parti necessarie del DOM reale invece dell'intero DOM. Questo meccanismo, ci permette di capire cosa deve essere aggiornato nel DOM e ottimizzare i vari passi, così che il browser non deve eseguire sempre gli stessi passi a ogni aggiornamento. Questo processo è chiamato riconciliazione. Questo approccio attiva l'API dichiarativa di React, a React viene detto in quale stato si vuole che la UI sia ed esso assicura che il DOM reale combaci lo stato. Questo processo astrae la manipolazione, gestione degli eventi e aggiornamento manuale del DOM. Tutto ciò porta come benefici migliori prestazioni di navigazione al browser, visto che non c'è bisogno di aggiornare l'intero DOM a ogni piccolo cambiamento dello stato.

3.8 CICLO DI VITA DEI COMPONENTI

Il ciclo di vita, dei componenti in React, ha tre fasi: Mounting, Updating e Unmounting. I metodi del ciclo di vita dei componenti React possono essere usati da componenti implementati tramite classi.

3.8.1 MOUNTING

Nella fase di Mounting i componenti vengono inseriti nel Virtual DOM. In questa fase è possibile usare 4 metodi per montare un componente:

- `constructor()`: è il metodo chiamato quando il componente è inizializzato e qui è consigliato di inizializzare lo stato del componente;
- `getDerivedStateFromProps()`: è il metodo chiamato poco prima di renderizzare l'elemento nel DOM, esso ritorna con i cambiamenti dello stato;
- `render()`: questo metodo renderizza i componenti JSX nel DOM;

- `componentDidMount()`: questo metodo è chiamato dopo che il componente è stato renderizzato.

3.8.2 UPDATING

Un componente quando ha un cambiamento di stato, React ha 5 metodi che sono chiamati mentre un componente si aggiorna:

- `getDerivedStateFromProps()`;
- `shouldComponentUpdate()`: questo metodo è usato quando si vuole decidere se aggiornare lo stato del componente oppure no, esso ritorna un valore booleano;
- `render()`;
- `getSnapshotBeforeUpdate()`: questo metodo è chiamato prima dell'update, con esso è possibile controllare lo stato precedente prima dell'update;
- `componentDidUpdate()`: esso è chiamato dopo che il componente è stato aggiornato.

3.8.3 UNMOUNTING

La fase finale di un componente è unmounting, questa fase è usata quando un componente è rimosso dal DOM, esiste solo un metodo usato da React:

- `componentWillUnmount()`: da qui è possibile liberare varie risorse usate in precedenza dal componente come chiamate ad API oppure liberare la cache.
-

3.9 REACT HOOK

I metodi che sono stati elencati in precedenza possono essere usati se e solo se i componenti sono implementati tramite le classi, quindi non possono essere usati in componenti implementati tramite funzioni, perché essi non possiedono uno stato e quindi non possono avere quei metodi. Esistono delle funzioni speciali chiamate hook che estendono le capacità dei componenti funzionali dando la possibilità a essi di avere gestione dello stato con i metodi che permettono la gestione del ciclo di vita. Le funzioni basiche sono:

- `useState()`: permette di dichiarare uno stato nel componente;
- `useEffect()`: usato per catturare manualmente i cambiamenti del DOM;

- `useContext()`: usato assieme con le API Context di React. Quando il React Context provider aggiorna, questo hook farà scattare la renderizzazione del componente con gli ultimi valori di contesto.

3.10 REACTJS DESIGN PATTERN

3.10.1 FUNZIONI STATELESS

Le funzioni stateless sono semplici funzioni Javascript. Quando si crea un componente in React di solito si costruisce una classe che estende da `React.Component` o `React.PureComponent`. Questi si occuperanno di gestire lo stato del componente per te, ma se si sta creando un semplice componente, questo può degradare le prestazioni del software. Le funzioni stateless non memorizzano alcuno stato, ma possono renderizzare un componente.

3.10.2 CONDITIONAL RENDERING

Questo pattern è usato quando si vuole renderizzare qualche componente JSX in base allo stato. Per esempio, se qualcosa è attivo, si renderizza il componente X, altrimenti si renderizza il componente Y. Si può avere questa funzionalità tramite il conditional rendering applicandolo ai componenti coinvolti.

3.10.3 CONTROLLED COMPONENTS

Un componente controllato è un componente dove Reactjs gestisce aggiornando il DOM per te. Quindi se si modifica un valore a dei componenti non controllati, esso verrà memorizzato all'interno del DOM reale e successivamente verrà aggiornato interamente. Questo può generare un degrado delle prestazioni se ci sono solo dei piccoli cambiamenti.

3.11 ROUTING IN REACT

Il routing è la capacità di muoversi tra diverse parti di un'applicazione quando un utente inserisce un URL, o clicca un elemento (link, pulsante, icona, immagine) all'interno dell'applicazione. Una rotta lato client accade quando è gestita internalmente da javascript, per esempio quando l'utente clicca su un link, l'URL cambia ma la richiesta non è inviata al server. Il cambiamento del URL si concretizzerà nel cambiamento di stato dell'applicazione. Per finire il cambiamento di stato si finalizzerà con il cambiamento della view della pagina web. Questo può essere come ad esempio, la renderizzazione di un nuovo componente, o anche una richiesta dal server per qualche dato che l'applicazione trasformerà in elementi HTML.

È da notare che l'intera pagina non verrà aggiornata quando si usa il routing lato client. Ci saranno solamente degli elementi all'interno dell'applicazione che cambieranno.

Le librerie per gestire il routing in questo progetto sono:

- react-router: il nucleo della libreria;
- react-router-dom: una variazione della libreria pensata per essere usata nello sviluppo delle web application.

3.11.1 ROUTERS

Il package react-router include un numero di routers che possono essere usati a seconda dal tipo di piattaforma target. Per applicazioni basate sul browser è stato utilizzato il componente router BrowserRouter. Il BrowserRouter è usato per applicazioni che hanno un server dinamico che conosce come gestire qualsiasi tipo di URL. Utilizzando questo componente è possibile impostare dappertutto delle nuove rotte all'interno del sorgente dell'app.

3.11.2 HISTORY

Ogni router crea un oggetto history che usa per tracciare la locazione corrente e di renderizzare l'applicazione nel caso in cui la corrente locazione cambiasse. L'oggetto contiene un certo numero di proprietà che possono essere utilizzate per cambiare, controllare e gestire la rotta corrente.

3.11.3 ROUTES

Il componente Route è uno dei più importanti nel package react-router. Il suo compito è quello di renderizzare l'appropriata interfaccia utente quando la locazione corrente combacia con il path della rotta. Il path è una proprietà del componente Route che descrive il nome del percorso che la rotta dovrebbe combaciare per mostrare il componente associato.

Questo componente fornisce tre proprietà che possono essere usate per determinare quale componente visualizzare:

- component: definisce l'elemento React che sarà restituito da Route quando il path combacerà;
- render: fornisce la capacità per la renderizzazione inline, permettendo di passare ulteriori proprietà all'elemento da renderizzare;
- children: questa proprietà è simile a render, l'unica differenza è che l'elemento definito da child è restituito a prescindere per tutti i percorsi.

Con questo tipo di componente è possibile definire anche delle rotte nidificate così da definire l'URL che riflette la particolare struttura nidificata. In altre parole il routing nidificato ci aiuta a renderizzare i componenti delle sotto-rotte. Così si ha l'abilità di visualizzare più informazioni che vengono mostrate dinamicamente senza il cambiamento completo della pagina corrente. Questo permette di avere una web app graficamente più organizzata e scalabile. È anche possibile definire delle rotte protette, una determinata rotta dovrebbe essere accessibile solo se l'utente ha effettuato il login, altrimenti l'utente dovrebbe essere indirizzato alla pagina di login. È possibile realizzare quest'ultima caratteristica tramite il conditional rendering.

3.11.4 SWITCH

Il componente Switch è usato per avvolgere multipli componenti Route. Serve per raccogliere solo la prima rotta che combacia tra tutte le altre annidate in esso.

3.11.5 LINK

Il package react-router contiene anche il componente Link che è usato per navigare le differenti parti di un'applicazione tramite collegamenti ipertestuali. È simile all'elemento anchor di HTML, l'unica differenza che c'è tra loro è che Link non ricarica la pagina, piuttosto cambia l'UI. Usando gli anchor tag si richiederebbe che la pagina venga ricaricata per caricare la nuova UI. Quando il componente Link è cliccato, viene anche aggiornato l'URL.

3.12 EXPRESS.JS

Express.js è un web application framework open source per Node.js. È usato per la progettazione e l'implementazione di web application in un modo rapido e semplice. Per utilizzare questo framework si richiede solamente la conoscenza di javascript, quindi permette agli sviluppatori di sviluppare web app e API senza troppi sforzi.

È possibile progettare con Express.js web application di tipo, single-page, multi-page o di tipo ibrido. Express.js è un framework leggero e aiuta a organizzare le web application sul lato server in una architettura più orientata al pattern MVC. Quindi il framework gestisce tutta la parte backend gestendo routing, sessioni, richieste http, questione degli errori e altro, senza avere a che fare con tanti elementi complessi che servono per la progettazione di una API efficiente e aggiunge altre caratteristiche e funzionalità a Nodejs.

Node.js è un framework per web app e servizi real-time e ha la capacità di gestire migliaia di richieste effettuate da differenti client, cosa che non era possibile con PHP. Questa tipologia di software, oggigiorno, sta diventando sempre più comune. Un esempio di app real-time è una app live-chat in cui sono coinvolti migliaia di utenti e l'interazione real-time può essere supportata facilmente con Node.js affiancato da Express.js. Express.js è un framework che riduce il tempo di sviluppo del software e in contemporanea permette di sviluppare software robusti ed efficienti.

3.12.1 CARATTERISTICHE DI EXPRESS.JS

- Sviluppo lato server più veloce: Express.js fornisce molte caratteristiche usate in Node.js nella forma di funzioni che possono essere prontamente usate dappertutto nel programma. Questo rimuove il bisogno di scrivere codice per diverse ore, risparmiando tempo;
- Middleware: un middleware è una parte del programma che ha accesso al database, richieste client e ad altri middleware. Perlopiù è responsabile per l'organizzazione sistematica di funzioni differenti di Express.js;
- Routing: Express.js fornisce un meccanismo di routing altamente avanzato il quale aiuta a preservare lo stato della web app con l'aiuto dei loro URL;
- Templating: Express.js fornisce dei templating engine che permettono agli sviluppatori di progettare contenuti dinamici sulle pagine web sviluppando dei template HTML sul lato server;
- Debugging: il debugging è cruciale per lo sviluppo di web application. Express.js rende il debugging più facile fornendo un meccanismo di debugging che ha l'abilità di localizzare l'esatta parte della web application che ha un bug;

3.12.2 VANTAGGI DI EXPRESS.JS

- Express.js è uno strumento molto popolare per via dei seguenti vantaggi nel suo uso:
- È facile da imparare perché molti sviluppatori front end sono già familiari con javascript. Così loro non devono imparare un nuovo linguaggio di programmazione per imparare a usare Express.js;
- Questo rende lo sviluppo backend molto più semplice per gli sviluppatori front end usando Express.js;
- Uno sviluppatore web può usare javascript come singolo linguaggio sia per lato client, sia per lato server. Quindi lo sviluppatore non ha bisogno di imparare o di usare qualsiasi altro linguaggio di programmazione per il lato server;
- Il codice Javascript è interpretato attraverso il Google V8 javascript Engine con Node.js. Quindi il codice è interpretato velocemente e facilmente in una maniera efficace;
- Express.js è semplice da personalizzare e usare in base ai propri bisogni;
- Express.js fornisce un flessibile modulo middleware. È principalmente usato per eseguire dei task extra sui responsi e richieste da parte di un client.

3.12.3 ROUTING IN EXPRESS.JS

Per routing si definisce a come l'endpoint di un'applicazione (URL) risponde alla richiesta del client. Per definire delle rotte è possibile usare i metodi forniti da Express.js dall'oggetto app che corrispondono ai metodi http. Per esempio app.get() è usato per gestire le richieste GET and app.post() è usato per gestire le richieste POST. Si può anche usare app.all() per gestire tutti i metodi http e app.use() per specificare un middleware come funzione callback.

Questi metodi in cui è specificato una rotta è anche specificato una funzione handler chiamata quando l'applicazione riceve una richiesta alla rotta e al metodo http specifico. In altre parole, l'applicazione attende delle eventuali richieste che combaciano la rotta e il metodo specifico e quando viene intercettata una corrispondenza, viene chiamata la specifica funzione handler.

Le rotte possono avere più di una funzione handler come argomento, quando si hanno multiple funzioni è importante chiamare il metodo next() in ognuna di esse, per passare il controllo alla successiva funzione.

Una rotta può anche avere dei parametri, nello specifico le rotte con parametri sono chiamate segmenti URL. Sono usate per catturare i valori specificati alla loro posizione nell'URL. I valori catturati sono memorizzati nell'oggetto req.params.

È possibile inviare un responso al client tramite i metodi dell'oggetto res, terminando il ciclo di vita del processo richiesta-responso. Se nessuno dei metodi di res viene chiamato dal gestore della rotta, la richiesta del client sarà lasciata in attesa.

Con la classe express.Router è possibile creare dei rotte handler modulari e montabili. Un'istanza Router è un completo middleware ed è anche un sistema di routing, per questa ragione è spesso riferita come mini-app.

3.13 WEBPACK

Webpack è definito come un modulo bundler statico che usa applicazione javascript in esso. Viene creato un grafico delle dipendenze che può essere usato per impacchettare tutti i moduli javascript in un solo singolo modulo siccome tutti i moduli javascript sono interdipendenti tra di loro. Per usare correttamente webpack bisogna capire alcuni importanti concetti:

- Entry: per il modulo webpack, gli entry point è il modulo da cui si parte per comporre il tipo di grafico delle dipendenze. Tutti gli entry point e moduli sono inclusi in questo grafico. Di solito l'entry point è il file: index.js. Anche un modulo separato può essere specificato nel file di configurazione di webpack;
- Output: è un altro tipo di proprietà che indica dove i bundle saranno memorizzati e quali nomi avranno i file di output. Gli altri file possono essere immagini oppure qualsiasi altro tipo di file;
- Loaders: ci sono due tipi di file che sono supportati e questi sono file JSON e file javascript. Per supportare qualsiasi altro tipo di file il webpack usa i loaders così che esso può convertirli in un formato valido per considerarli come moduli;
- Plugins: i plugins sono l'alternativa ai loaders, essi sono più potenti dei loaders ed eseguono diverse funzionalità che i loaders non possono eseguire. I plugins sono usati per gestire l'asset, minimizzare il bundle e ottimizzare i blundle così che può essere usato in un modo migliore;
- Modalità: ci sono due categorie di codice sorgente, uno è per la produzione e l'altro è per lo sviluppo. Il modulo è usato per cambiare la modalità a produzione, sviluppo oppure ad altre modalità. Di default, la modalità è impostata a produzione.

Siccome il webpack è usato per compilare i moduli javascript. Dopo l'installazione, l'utente può interagire con esso tramite API o CLI.

Ci sono diversi fattori che rendono importante l'uso di webpack, alcuni di questi sono:

- Automazione: tutto il lavoro di automazione è direttamente fatto da webpack. Non c'è alcun bisogno di aggiungere librerie javascript nell'header HTML. Webpack può essere usato per fare tutti questi compiti e aiuta agli sviluppatori a risparmiare tempo;

- **Velocità di caricamento:** quando qualsiasi file di script è caricato nella pagina web, può impiegarci molto tempo per essere visualizzata, soprattutto se la pagina web è abbastanza complessa. Webpack aiuta a ridurre il costo di caricamento di tutti i moduli, visto che questi possono essere impacchettati in un singolo modulo. Quindi il web server, invece di ottenere tutti i moduli singolarmente, ne ottiene solo uno in cui sono contenuti tutti gli altri;
- **Caricamento script importanti:** quando una normale web application viene eseguita, tutti i moduli javascript sono caricati, creano un degrado delle prestazioni non necessario, ma nel webpack solo i moduli necessari sono caricati, migliorando le prestazioni del web server. Il webpack permette di dividere il codice da eseguire in più parti, così gli script vengono caricati su richiesta dell'applicazione;
- **Problemi di dipendenza:** i problemi di dipendenza sono eliminati mentre si usa il webpack. Quindi le dipendenze tra le librerie e gli script in tale maniera sono ridotte di molto;
- **Maggiore velocità di sviluppo:** il webpack offre dei moduli che permettono di migliorare la produttività di sviluppo.

Il webpack è un tipo di applicazione che è usato per compilare i moduli javascript. Usando il webpack l'utente può impacchettare diversi moduli in uno singolo. Il grafico delle dipendenze viene creato basandosi su tutti i moduli. Le dipendenze dei moduli possono essere rimosse facilmente tramite webpack e permette di ridurre il tempo di sviluppo.

React fa affidamento su webpack perché sfrutta queste sue caratteristiche:

- Compilazione di codice da JFX a javascript;
- Compilazione di codice da javascript con sintassi ES6 a versioni javascript supportati dai browser;
- Gestione delle importazioni di moduli nel codice sorgente e impacchettamento di questi in un unico modulo da usare in caso di produzione;
- Esecuzione di varie ottimizzazioni nella fase di impacchettamento.

3.14 FETCHING

3.14.1 FUNZIONE FETCH

Javascript può inviare richieste al server e caricare nuove informazioni quando ce n'è bisogno. Per esempio, possiamo usare una richiesta network per:

- Inviare un comando;
- Caricare informazioni per l'utente;
- Ricevere gli ultimi aggiornamenti dal server.

Il tutto accade senza dover ricaricare la pagina web. Esiste un termine specifico in javascript che sta a indicare questa funzionalità: **AJAX (Asynchronous JavaScript And XML)**. Non bisogna per forza usare XML, il termine proviene dal periodo in cui XML veniva usato al posto di JSON. Il modo in cui javascript esegue il fetching è tramite la funzione globale `fetching` che prende come parametri un url e delle opzioni e ritorna una promessa. Nella promessa è possibile ottenere il responso della richiesta. La funzione `fetch` è di questo tipo: `let promise = fetch(url, [options])`.

Se non vengono fornite opzioni, il fetching viene interpretato con un semplice GET request che scarica il contenuto del url passato come parametro.

Quando si ottiene il responso, si può controllare sia lo status, sia l'headers del messaggio http, ma senza avere il body del responso. È possibile chiamare i seguenti metodi:

- `response.text()`: legge il responso come testo;
- `response.json()`: esegue il parsing del responso come JSON;
- `response.formData()`: restituisce il responso come oggetto `FormData`;
- `response.blob()`: restituisce il responso come `Blob`;
- `response.arrayBuffer()`: restituisce il responso come `ArrayBuffer`.

La funzione `fetch` può essere usata con altre funzioni callback e anche con le keywords `async/await`.

3.14.2 AXIOS

Axios è una libreria javascript basata su promesse client http per il browser e Node.js. Axios rende più facile inviare richieste http asincrone a un endpoint REST e rende più facile le operazioni CRUD. Può essere integrato con librerie come React.

Ci sono diversi metodi per generare richieste in axios:

- `axios.get(url, [config]):` per effettuare una richiesta GET al server;
- `axios.delete(url, [config]):` per effettuare una richiesta DELETE al server;
- `axios.post(url, [data, [config]]):` per effettuare una richiesta POST al server.

Quando si invia una richiesta a un server, axios ritorna un responso. L'oggetto Axios del responso consiste in:

- `data:` il payload ritornato dal server;
- `status:` il codice http restituito dal server;
- `statusText:` il messaggio status http restituito dal server;
- `headers:` gli headers inviato dal server;
- `config:` la richiesta originale di configurazione;
- `request:` l'oggetto richiesta.

3.15 TECNOLOGIE

3.15.1 JAVASCRIPT

Javascript è un linguaggio di programmazione, interpretato, dinamico, debolmente tipato, multi-paradigma e ad alto livello conforme alle specificazioni ECMAScript. Affiancato ad HTML e CSS è uno di capisaldi della programmazione Web. Infatti, tutti i Web Browser più importanti hanno un dedicato Javascript Engine così da eseguire il codice sul dispositivo utente. Una caratteristica importante di Javascript è quella di permettere la manipolazione DOM (Document Object Model). Anche se lo scopo primario di Javascript era quello di rendere le pagine Web dinamiche, oggi può essere utilizzato anche per lo sviluppo non Web, come ad esempio lo sviluppo di server.

3.15.2 TYPESCRIPT

Typescript è un linguaggio di programmazione sviluppato e mantenuto da Microsoft. Esso si può definire come un sovrainsieme di Javascript che aggiunge, opzionalmente, tipi statici. Esso è stato progettato e per permettere lo sviluppo di grandi applicativi. Una caratteristica particolare di Typescript è che i sorgenti scritti in questo linguaggio vengono compilati in Javascript, così è possibile notare e

correggere errori che possono essere ardui da rilevare dinamicamente. È anche possibile aggiungere varie opzioni al compilatore prima di convertire il sorgente in Javascript. Typescript può essere utilizzato per applicazioni lato client e lato server. Anche se esso è un sovrainsieme di Javascript, i framework sviluppati per quest'ultimo, possono essere impiegati anche in Typescript.

3.15.3 REACTJS

React è una libreria open source per Javascript per lo sviluppo lato client. Con esso è possibile creare interfacce utente oppure componenti grafici. È mantenuto da Facebook e da sviluppatori individuali. Esso può essere considerato come una base per lo sviluppo di Web application lato client, però lo scopo del framework è quello di preoccuparsi per la gestione dello stato dell'applicativo e di renderizzare lo stato nel DOM. Quindi lo sviluppo di applicazioni React, di solito richiede l'uso di librerie aggiuntive.

3.15.4 JSX

JSX è un'estensione alla sintassi del linguaggio Javascript. Esso fornisce un modo di strutturare i componenti usando una sintassi familiare ad HTML.

3.16

3.16.1 MATERIAL UI

Framework open source per ReactJS che fornisce componenti basati su Material Design creato da Google.

3.16.2 CSS

CSS è un linguaggio di stile usato per descrivere la presentazione di un documento scritto in un markup language come HTML. Esso è progettato per separare la presentazione e contenuto, includendo layout, colori e stili di testo. CSS è uno degli strumenti più utilizzati per lo sviluppo web

3.16.3 NODEJS

NodeJS è un ambiente Javascript runtime, open source, cross-platform che è eseguito nel V8 engine ed esegue codice Javascript fuori dal Web Browser. Grazie a questo ambiente è possibile creare server utilizzando il linguaggio Javascript, così è possibile impiegare un solo linguaggio di programmazione per lo sviluppo lato client e lato server delle web application. Esso fornisce a Javascript alcuni strumenti che l'ECMAScript non definisce come: file system I/O, networking e data streams.

3.16.4 EXPRESS

Express è un framework open source per lo sviluppo di web application lato server e di API. È de facto lo standard per lo sviluppo di server per NodeJS.

3.16.5 AXIOS

Axios è una libreria che permette la comunicazione tra client e server basata su promesse HTTP.

3.16.6 NPM

Npm (Node Package Manager) è un package manager per il linguaggio di programmazione Javascript ed è il default package manager per l'ambiente NodeJs.

3.16.7 POSTGRESQL

PostgreSQL è un RDBMS open source che enfatizza estendibilità e SQL compliance. Esso è caratterizzato da transazioni con proprietà di atomicità, consistenza, isolazione e durabilità (ACID), viste automaticamente aggiornabili, viste materializzabili, triggers, foreign key e stored procedure. È stato progettato per gestire un certo carico di lavoro, da singole macchine a data warehouse or servizi web con tanti utenti concorrenti.

3.16.8 VISUAL STUDIO CODE

Visual Studio Code è un editor open source che permette lo sviluppo di software in vari linguaggi, sviluppato da Microsoft che include molte caratteristiche come: debugging, completamento codice e varie altre estensioni.

3.16.9 GIT

Git è un software di versionamento che permette di tener traccia dei cambiamenti di un insieme di files. Esso è impiegato per la coordinazione e la collaborazione di un team di sviluppatori.

3.16.10 GITLAB

GitLab è uno strumento basato su web che permette di monitorare il ciclo di vita di un software, fornendo un repository per Git.

CAPITOLO: 4 TESTING

4.1 DESCRIZIONE ATTIVITÀ

Lo scopo di testing è stato quello di verificare se i vari casi d'uso corrispondessero a ciò che è stato richiesto nell'analisi dei requisiti. Quindi tutte le funzionalità sono state testate in modo tale da verificare se ci fossero eventuali incongruenze, eventuali bug all'interfaccia utente ed eventuali bug di tipo logici. Nel caso in cui problemi del genere si sono manifestati, sono stati prontamente risolti. All'interno dei componenti grafici sono stati inseriti vari valori appartenenti a diverse classi di equivalenza per verificare se lo scenario principale e qualsiasi sotto-scenario di ogni caso d'uso funzionassero correttamente e se qualsiasi eccezione descritta nell'analisi dei requisiti venisse gestita correttamente.

4.2 TESTING DEL SISTEMA

Test ID	1	
Test Name	Visualizza somministrazioni terapia farmacologica test	
Test Description	Si testa se la web app permette di visualizzare correttamente le somministrazioni di una terapia farmacologica	
Input	Risultato desiderato	Risultato ottenuto
Seleziona terapia	Mostra misurazioni in forma grafica e tabellare	Mostra misurazioni in forma grafica e tabellare
Nessuna terapia esistente	Mostra errore	Mostra errore
Seleziona terapia ma senza connessione a internet	Mostra errore	Non mostra nulla
Location	Mozilla firefox versione 88.0.1 (64-bit)	
Notes		

Test ID	2	
Test Name	Aggiungi Paziente test	
Test Description	Si testa se la web app aggiunge correttamente un paziente e se vengono gestiti correttamente gli errori	
Input	Risultato desiderato	Risultato ottenuto
Campi corretti	Aggiunge il nuovo paziente e mostra "aggiungi-modifica care plan"	Aggiunge il nuovo paziente e mostra "aggiungi-modifica care plan"
Campi corretti ma senza interventi chirurgici	Aggiunge il nuovo paziente e mostra "aggiungi-modifica care plan"	Aggiunge il nuovo paziente e mostra "aggiungi-modifica care plan"
Campi corretti ma senza vaccinazioni	Aggiunge il nuovo paziente e mostra "aggiungi-modifica care plan"	Aggiunge il nuovo paziente e mostra "aggiungi-modifica care plan"
Campi corretti ma senza allergie	Aggiunge il nuovo paziente e mostra "aggiungi-modifica care plan"	Aggiunge il nuovo paziente e mostra "aggiungi-modifica care plan"
Campi corretti ma senza patologie pregresse	Aggiunge il nuovo paziente e mostra "aggiungi-modifica care plan"	Aggiunge il nuovo paziente e mostra "aggiungi-modifica care plan"
Codice fiscale già inserito	Mostra messaggio di errore	Mostra messaggio di errore
Uno o più campi obbligatori vuoti	Mostra messaggio di errore	Mostra messaggio di errore
Campi corretti ma nessuna connessione a internet	Mostra messaggio di errore	Non accade nulla
Location	Mozilla firefox versione 88.0.1 (64-bit)	
Notes		

Test ID	3	
Test Name	Inserisce-Modifica care plan test	
Test Description	Si testa se la web app inserisce/modifica il care plan di un paziente e se gestisce correttamente i vari errori	
Input	Risultato desiderato	Risultato ottenuto
Inserisce campi correttamente	Inserisce il care plan e mostra messaggio di successo	Inserisce il care plan e mostra messaggio di successo
Aggiorna campi correttamente	Aggiorna il care plan e mostra messaggio di successo	Aggiorna il care plan e mostra messaggio di successo
Inserisce care plan senza una o più monitorazioni	Inserisce il care plan e mostra messaggio di successo	Inserisce il care plan e mostra messaggio di successo
Aggiorna care plan senza una o più monitorazioni	Aggiorna il care plan e mostra messaggio di successo	Aggiorna il care plan e mostra messaggio di successo
Inserisce care plan senza terapie farmacologiche	Inserisce il care plan e mostra messaggio di successo	Inserisce il care plan e mostra messaggio di successo
Aggiorna care plan senza terapie farmacologiche	Aggiorna il care plan e mostra messaggio di successo	Aggiorna il care plan e mostra messaggio di successo
Aggiorna care plan rimuovendo una o più terapie farmacologiche	Aggiorna il care plan e mostra messaggio di successo	Aggiorna il care plan e mostra messaggio di successo
Uno o più campi vuoti	Mostra messaggio di errore	Mostra messaggio di errore
Nessuna connessione a internet	Mostra messaggio di errore	Errore a tempo di esecuzione
Location	Mozilla firefox versione 88.0.1 (64-bit)	
Notes		

Test ID	4	
Test Name	Aggiunge somministrazione terapia farmacologica test	
Test Description	Si testa se la web app permette di aggiungere una somministrazione di una terapia e se gestisce correttamente eventuali errori	
Input	Risultato desiderato	Risultato ottenuto
Campi corretti	Aggiunge somministrazione e mostra messaggio di successo	Aggiunge somministrazione e mostra messaggio di successo
Campi corretti ma senza note	Aggiunge somministrazione e mostra messaggio di successo	Aggiunge somministrazione e mostra messaggio di successo
Campi corretti ma con somministrazione non avvenuta con successo	Aggiunge somministrazione e mostra messaggio di successo	Aggiunge somministrazione e mostra messaggio di successo
Campi corretti ma senza data	Mostra errore	Mostra errore
Nessuna connessione a internet	Mostra errore	Mostra errore
Location	Mozilla firefox versione 88.0.1 (64-bit)	
Notes		

Test ID	5	
Test Name	Aggiunge misurazione parametro clinico test	
Test Description	Si testa se la web app permette di aggiungere una misurazione di un parametro clinico e se gestisce correttamente eventuali errori	
Input	Risultato desiderato	Risultato ottenuto
Campi corretti	Aggiunge misurazione e mostra messaggio di successo	Aggiunge misurazione e mostra messaggio di successo
Campi corretti ma senza note	Aggiunge misurazione e mostra messaggio di successo	Aggiunge misurazione e mostra messaggio di successo
Campi corretti ma senza data	Mostra errore	Mostra errore
Campi corretti ma senza valore misurato	Mostra errore	Mostra errore
Nessuna connessione a internet	Mostra errore	Mostra errore
Location	Mozilla firefox versione 88.0.1 (64-bit)	
Notes		

Test ID	6	
Test Name	Visualizza misurazioni parametri clinici test	
Test Description	Si testa se la web app permette di visualizzare correttamente le misurazioni di un parametro clinico	
Input	Risultato desiderato	Risultato ottenuto
Seleziona parametro	Mostra misurazioni in forma grafica e tabellare	Mostra misurazioni in forma grafica e tabellare
Nessun parametro da monitorare	Mostra errore	Mostra errore
Seleziona parametro ma senza connessione a internet	Mostra errore	Non mostra nulla
Location	Mozilla firefox versione 88.0.1 (64-bit)	
Notes		

CAPITOLO: 5 CONCLUSIONI

Riassumendo, in questo documento è descritta la progettazione e l'implementazione di un applicativo per la gestione e il monitoraggio di piani terapeutici condivisi, per migliorare la comunicazione tra medico e paziente implementato con tecnologie impiegate per lo sviluppo web.

Grazie agli enormi passi fatti nel settore informatico è stato possibile impiegare questi progressi e adattarli nel campo dell'Health Care per migliorare le cure mediche, migliorare la prevenzione di malattie e patologie e monitorare lo stato di salute di un paziente. Oggigiorno esistono in commercio molti strumenti, anche per uso domestico come gadget indossabili, che permettono il monitoraggio della attività fisica giornaliera e dei parametri vitali come il battito cardiaco, pressione sanguigna e temperatura corporea. Sarebbe stato impensabile sviluppare questi dispositivi anni fa per colpa della tecnologia limitata.

Lo shared care plan verrà utilizzato come applicativo base per lo sviluppo di un progetto più complesso e con più funzionalità. Siccome la web application è un prototipo, sarà possibile estendere facilmente le proprie funzionalità, aggiungerne delle nuove, adattarlo per ulteriori dispositivi e grazie ai micro-servizi sarà possibile integrarlo con nuove tecnologie.

Nel periodo del tirocinio ho avuto la possibilità di studiare nuovi strumenti e nuove tecniche di programmazione per la progettazione e lo sviluppo software e di impiegare queste nuove conoscenze per la realizzazione di un applicativo che può essere utilizzato nel mondo professionale medico. Quindi in questo periodo, ho avuto modo di affacciarmi nel mondo del lavoro e di conoscere questo ambiente mettendo da parte un po' di esperienza lavorativa.

5.1 SVILUPPI FUTURI

5.1.1 FUNZIONALITÀ AGGIUNTIVE

È possibile aggiungere ulteriori funzionalità allo shared care plan, in modo tale da migliorare il supporto e l'assistenza del personale medico fornito al paziente:

Una prima funzionalità che è possibile implementare è la composizione di un piano nutrizionale per il paziente, così il personale medico potrà definire un piano nutrizionale specifico al paziente, basandosi sui bisogni di quest'ultimo, definire periodicamente obiettivi nutrizionali e monitorare i progressi;

Una seconda funzionalità che è possibile implementare è la composizione di un piano di attività fisica, quindi il personale medico potrà seguire e monitorare un determinato paziente, che ha avuto un qualche tipo di infortunio oppure con un problema motorio, fino alla sua completa riabilitazione;

Una terza funzionalità che è possibile implementare è la gestione delle prescrizioni di un piano terapeutico. Un Piano Terapeutico è una particolare prescrizione necessaria per alcuni farmaci che possono essere prescritti solo da Centri Specialistici autorizzati dalla Regione. Quindi è possibile facilitare la prescrizione di quest'ultimi, facilitando la comunicazione tra corpo medico e paziente.

Una quarta funzionalità che è possibile implementare sono le notifiche push. Utilizzando npm è possibile scaricare e usare uno dei tanti dei microservizi come: react-push-notification, che permette, allo sviluppatore, di implementare le notifiche push. Esse possono essere utilizzate per notificare al medico o all'infermiere di eseguire una determinata misurazione o di somministrare un determinato farmaco a un paziente specifico, evitando così possibili dimenticanze.

5.1.2 RIADATTAMENTO PER ALTRI DISPOSITIVI

Lo shared care plan è stato pensato per dispositivi con uno schermo di grande dimensione, come PC e notebook. Oggigiorno è possibile accedere e usufruire delle web application anche tramite dispositivi con uno schermo di limitata dimensione, come smartphone e tablet. È possibile rendere e adattare i componenti react in modo tale da renderli dinamici e reattivi a seconda dello schermo su cui vengono renderizzati, in modo tale da rendere la web application mobile user friendly.

Per layout dinamico e reattivo si intende che viene utilizzato altro codice logico javascript per renderizzare dinamicamente una differente struttura dei componenti react basandosi sulla dimensione dello schermo, per esempio tramite varie tecniche di conditional rendering, è possibile definire se un determinato componente deve essere renderizzato oppure no, oppure è possibile progettare differenti tipi di componenti, a seconda del tipo dispositivo in cui devono essere renderizzati e tramite controlli javascript si farà renderizzare un componente o un altro basandosi dal tipo di dispositivo.

Successivamente si utilizzano delle proprietà di CSS per alterare la struttura dei componenti in modo reattivo, in questo caso è possibile modificare la dimensione e anche la posizione dei componenti, questo è possibile grazie all'uso delle media query in CSS, esse sono delle query che permettono di applicare differenti stili di design per differenti tipi di dispositivi. Le query possono essere usate per controllare molti elementi come:

- larghezza e altezza del viewport;
- larghezza e altezza del dispositivo;
- orientazione del dispositivo (se esso è orientato verticalmente o orizzontalmente);
- risoluzione

Come risultato si avrà una web application, adattabile a seconda della dimensione dello schermo, facilmente scalabile a livello grafico senza sacrificare un design pulito per gli utenti mobile.

5.1.3 SVILUPPO NATIVO PER ANDROID E IOS

Nel caso in cui non si voglia percorrere la strada del riadattamento della web app per dispositivi mobile è possibile convertire l'applicativo, seppur con qualche modifica al codice, in modo tale che giri in modo nativo su dispositivi Android e IOS usando il framework React Native. React Native è un framework usato per lo sviluppo di applicazioni mobile usando il linguaggio javascript. Il framework usa i componenti UI, i vari tipi di costrutti e pattern design e tutto il ciclo di vita dei componenti usati in React standard. In React Native si possono utilizzare tutte le librerie che possono essere impiegate in React standard come Redux. Il linguaggio che permette la modellazione dei componenti usato da React Native è sempre JSX anche se i componenti base non sono elementi del DOM ma si usa sempre la stessa sintassi per definirli.

Tutte le dipendenze javascript continuano a essere gestite da npm e tracciate nel file package.json. Però qualsiasi dipendenza che fa affidamento su HTML e SVG non sarà possibile utilizzarle in React Native.

Quindi nel caso in cui si voglia intraprendere questa strada, bisognerà modificare i componenti base standard impiegati dal DOM, per esempio alcuni componenti come: <div>, <p>, <input> dovranno essere sostituiti con componenti nativi come: <Text>, <View>, <TextInput> e <Image>. Molti componenti progettati da terzi possono essere aggiunti tramite npm. Per modificare lo stile dell'applicativo in React Native non utilizza CSS ma vengono impiegati oggetti StyleSheet. Questi oggetti hanno una sintassi abbastanza simile a CSS e vengono applicati all'attributo style di un determinato componente grafico. Per l'implementazione delle rotte in un'applicazione mobile si può scegliere tra React Navigation e React Native Navigation. Con la prima opzione l'implementazione delle rotte viene effettuata esclusivamente in javascript, mentre con la seconda opzione si utilizzano le API native associate con IOS e Android.