Relazione di Ingegneria del Software

Kevin Gecchele (VR500239)
Christian Dalla Valle (VR500076)

Settembre 2025

Indice

1 Introduzione

1.1 Panoramica

La traccia scelta è la numero 3 relativa alla gestione di un sistema di rilevazioni e organizzazione di terapie per la glicemia per medici e pazienti.
Il lavoro è stato svolto da Kevin Gecchele (VR500239) e Christian Dalla Valle (VR500076) ed è stato diviso in modo che potesse essere il più equo possibile:

- Dalla Valle che si è occupato della gestione di errori, interfaccia grafica, debugging e sezioni secondarie
- Gecchele ha curato la realizzazione dei grafici, della relazione e delle immagini per essa
- in comune trovandosi in presenza è stata sviluppata la struttura, le idee, le sezioni principali di codice e la gestione dei file CSV per il salvataggio dei dati.

Come già anticipato sono stati usati file CSV per il salvataggio dei dati in modo da mantenerli ad ogni interazione con l'applicazione.
I dati su pazienti e medici vengono considerati già inseriti nel sistema e non gestiti dall'applicazione che invece li usa solo per l'autenticazione, l'identificazione del diabetologo assegnato al paziente e l'invio delle mail per i contatti diretti tra diabetologo e paziente.

1.2 Funzionalità

Diabetologo

- Login
- Logout
- Visualizzare lista pazienti
- Modificare scheda clinica pazienti
- Visualizzare rilevazioni glicemia pazienti **
- Visualizzare assunzioni farmaci pazienti **

- Visualizzare eventuali sintomi/patologie pazienti **
- Visualizzare eventuali altre terapie esterne pazienti **
- Assegnare terapie pazienti
- Modificare stato terapie pazienti *

Paziente

- Login
- Logout
- Contattare medico (client mail esterno)
- Visualizzare lista terapie
- Inserire altre terapie esterne concomitanti
- Inserire eventuali sintomi/patologie
- Inserire rilevazioni glicemia
- Inserire assunzioni farmaci
- Modifica rilevazioni glicemia *
- Ricevere alert dopo rilevazioni mancanti *

NOTE *: la funzione non è presente nel grafico dei casi d'uso perché considerabile come secondaria o come flusso alternativo di un altro caso d'uso (non rappresentati).
NOTE **: la funzione non è presente nel grafico dei casi d'uso perché incorporata e trattata insieme ad altre in *Monitorare dati paziente*.

Queste scelte sono state prese per rendere i grafici più leggibili e non farli espandere troppo in orizzontale rendendoli difficilmente analizzabili a causa delle dimensioni per farli stare nella pagina.

2 Funzionamento

2.1 Casi d'uso

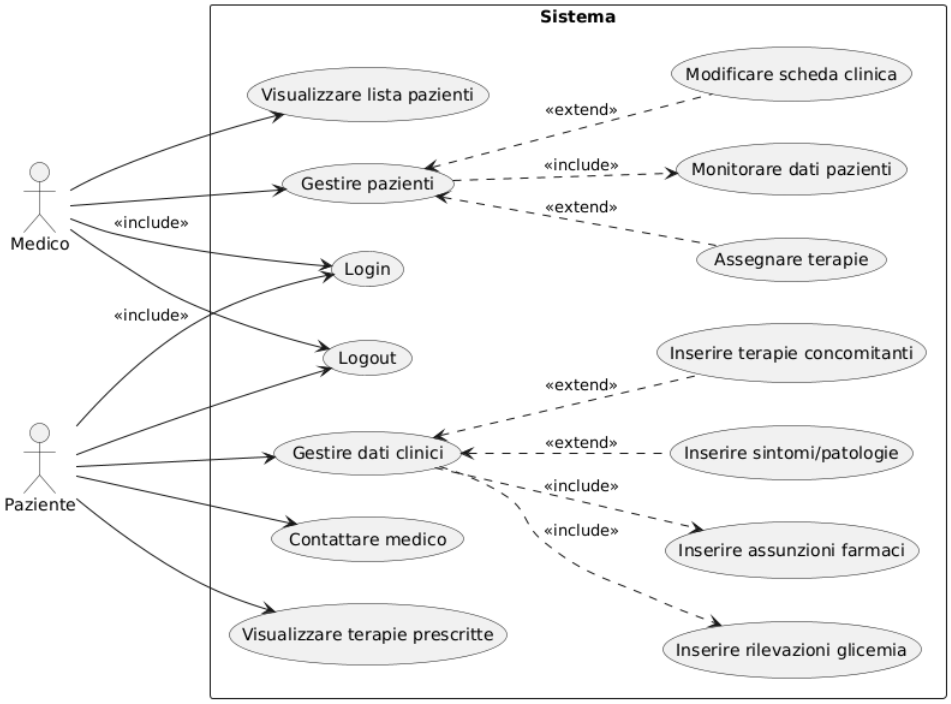

Figura 1: Casi d'uso

Sia medici che pazienti **devono** effettuare il *login* per accedere alle loro rispettive pagine. L'assegnazione alle pagine avviene automaticamente valutando le credenziali. Entrambi dopo aver eseguito l'accesso possono decidere di fare il *logout* per tornare alla pagina di login. Il grafico 1 non tiene conto di ritorni o flussi alternativi per questioni di leggibilità.
Per prima cosa vengono trattati i casi *login* e *logout* perché in comune tra gli attori, e successivamente nelle sottosezioni apposite i casi specifici di medici e pazienti.
NOTA: Per semplificare e rendere più leggibile i sequence diagram gli input e gli output consecutivi di un attore vengono raggruppati in uno solo.

Use Case: Login
Attori: diabetologo/paziente
Precondizioni: l'utente non ha effettuato l'accesso
Sequenza degli eventi: 1. Il caso inizia quando viene avviata l'applicazione o viene effettuato il logout 2. L'utente inserisce le sue credenziali d'accesso 3. L'utente clicca sul punsate per avviare la verifica (a) **Se** l'autenticazione va a buon fine viene effettuato l'accesso (b) **Se** l'autenticazione fallisce l'utente riceve un messaggio d'errore e riparte dal punto 1
Postcondizione: l'utente può accedere alla sua sezione nell'applicazione

Tabella 1: Specifiche di **Login**

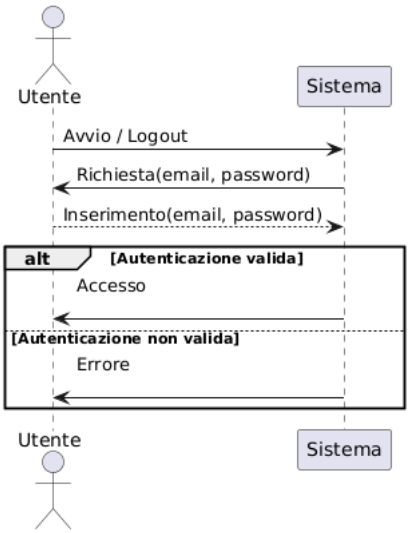

Figura 2: Sequence Diagram per **Login**

2.1.1 Casi d'uso del diabetologo

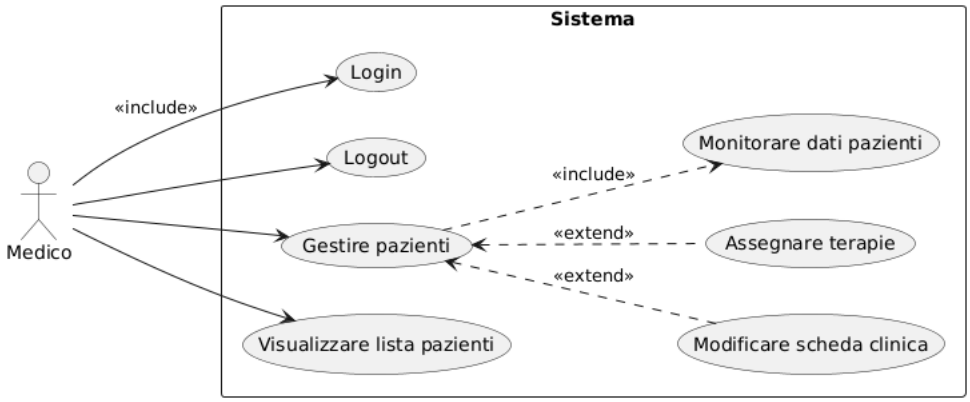

Figura 3: Casi d'uso del diabetologo

Per quanto riguarda i casi in cui è coinvolto esclusivamente il diabetologo, esso può visualizzare la lista dei suoi pazienti e, cliccando su uno tra loro, può accedere alla scheda a lui relativa. Per monitorare il paziente è necessario osservarne i dati, principalmente le sue rilevazioni dei livelli di glicemia e le assunzioni, poi se presenti anche sintomi, patologie e terapie esterne che sta seguendo, ma essendo che non è detto che il paziente fornisca quest'ultime informazioni per mancanza di questi eventi opzionali, esse non sono consultabili in ogni istanza del caso d'uso. Se necessario il diabetologo può assegnare terapie al paziente o modificarne la scheda clinica, ma anche queste operazioni sono opzionali.

Use Case: Monitorare dati pazienti
Attori: diabetologo
Precondizioni: il diabetologo ha effettuato l'accesso
Sequenza degli eventi: 1. Il caso inizia dopo che il diabetologo seleziona un paziente dalla lista 2. Il diabetologo viene portato sulla pagina del paziente, in cui sono presenti i valori della glicemia, le assunzioni, i sintomi, le patologie e le terapie esterne
Postcondizione: il diabetologo può consultare i dati per valutare eventuali terapie o modifiche alla scheda clinica del paziente

Tabella 2: Specifiche di **Monitorare dati pazienti**

Figura 4: Sequence Diagram per **Monitorare dati pazienti**

Use Case: Assegnare terapie
Attori: diabetologo
Precondizioni: il diabetologo si trova nella pagina delle terapie di un paziente
Sequenza degli eventi: 1. Il caso inizia dopo che il diabetologo seleziona la funzione "Aggiungi" nella pagina delle terapie 2. Il sistema gli richiede le informazioni tramite form 3. Il diabetologo seleziona inserisce le informazioni richieste (farmaco, assunzioni, quantità, indicazioni, inizio, fine) 4. Il sistema crea la terapia con le informazioni fornite
Postcondizione: la terapia è consultabile da diabetologo e paziente nella pagina delle terapie

Tabella 3: Specifiche di **Assegnare terapie**

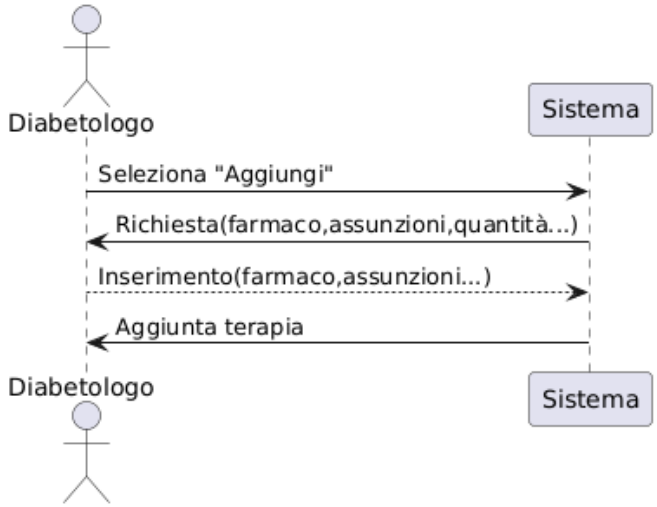

Figura 5: Sequence Diagram per **Assegnare terapie**

Use Case: Modificare scheda clinica
Attori: diabetologo
Precondizioni: il diabetologo ha selezionato un paziente
Sequenza degli eventi: 1. Il caso inizia dopo che il diabetologo clicca sulla sezione relativa alla scheda clinica del paziente 2. Il diabetologo modifica le sezioni dei fattori di rischio, patologie pregresse e comorbidità 3. Il diabetologo clicca sul pulsante "Aggiorna" 4. Il sistema aggiorna la scheda clinica
Postcondizione: la scheda risulta modificata nella pagina relativa al paziente

Tabella 4: Specifiche di **Modificare scheda clinica**

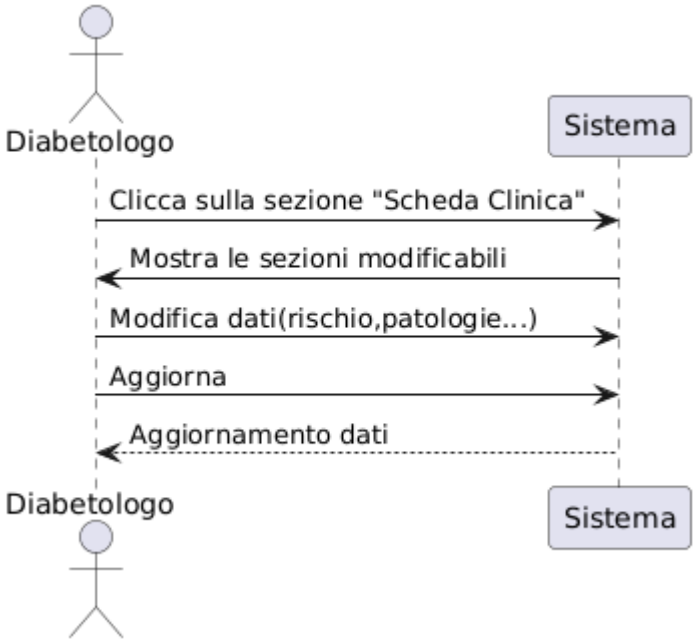

Figura 6: Sequence Diagram per **Modificare scheda clinica**

Visualizzare lista pazienti inizia quando il diabetologo effettua l'accesso e si trova nella pagina principale. La lista dei pazienti viene stampata sulla pagina e ogni elemento è selezionabile dal diabetologo cliccandoci e portandolo nella pagina del paziente.
Monitorare dati pazienti inizia quando il diabetologo seleziona dalla lista uno dei pazienti ed entra nella sua pagina. Lì vengono stampate tutte le informazioni a lui relative per rilevazioni (colorate a seconda dei range di valore), assunzioni, sintomi, patologie e terapie esterne. Il diabetologo può sfruttare questi dati per regolare o assegnare terapie al paziente.

2.1.2 Casi d'uso del paziente

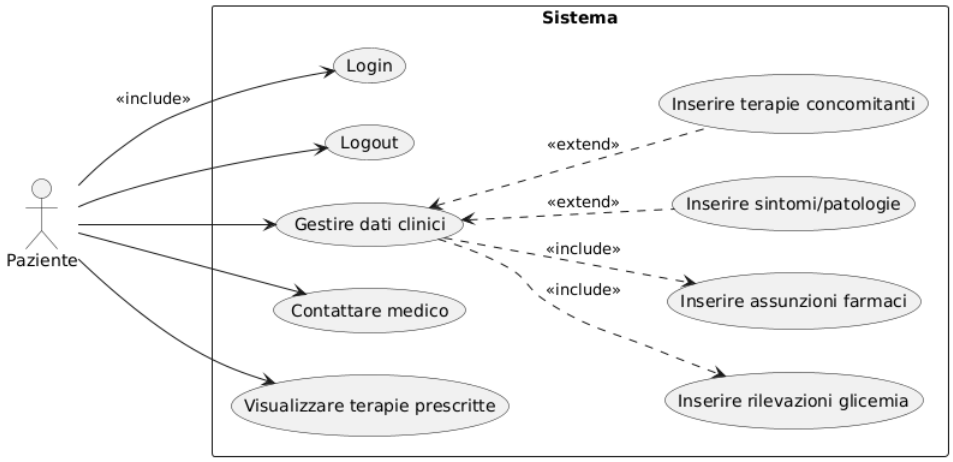

Figura 7: Casi d'uso del paziente

I casi d'uso relativi al paziente sono la possibilità di contattare il medico tramite collegamento con un sistema di email esterno, la visualizzazione delle terapie prescritte dal diabetologo e la gestione dei propri dati clinici riguardanti le rilevazioni glicemiche e le assunzioni di farmaci prescritti. Se si verificano eventuali sintomi, patologie o se il paziente sta seguendo una terapia esterna deve segnalarlo, ma si tratta di operazioni opzionali in quanto non è detto che si verifichino per ogni istanza del caso d'uso in cui sono inclusi.

Use Case: Contattare medico
Attori: paziente
Precondizioni: paziente ha effettuato l'accesso
Sequenza degli eventi: 1. Il caso inizia dopo che il paziente seleziona la funzione "Contatta" nella pagina principale. 2. Il sistema fa comparire una casella di testo 3. Il paziente scrive il messaggio che vuole inviare al diabetologo e preme INVIO 4. Il sistema apre il client email con la bozza della mail richiesta dal paziente 5. Il paziente clicca il punsante "Invia" 6. Il client invia la mail al diabetologo
Postcondizione: il messaggio viene inviato via mail al diabetologo che può leggerlo

Tabella 5: Specifiche di **Contattare medico**

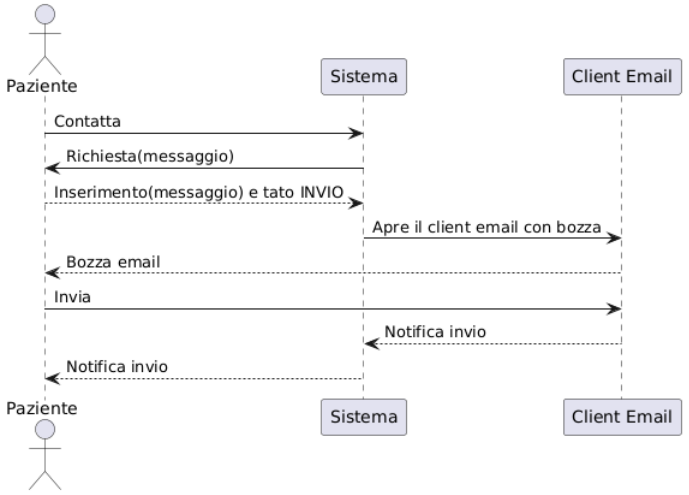

Figura 8: Sequence Diagram per **Contattare medico**

Use Case: Inserire assunzioni farmaci
Attori: paziente
Precondizioni: il paziente ha effettuato l'accesso
Sequenza degli eventi: 1. Il caso inizia dopo che il paziente seleziona la scheda relativa alle assunzioni 2. Il paziente immette data, ora, farmaco e quantità 3. Il paziente seleziona "Aggiungi" 4. Il sistema si aggiorna con l'assunzione
Postcondizione: l'assunzione risulta aggiunta ed è visibile anche dal diabetologo

Tabella 6: Specifiche di **Inserire assunzioni farmaci**

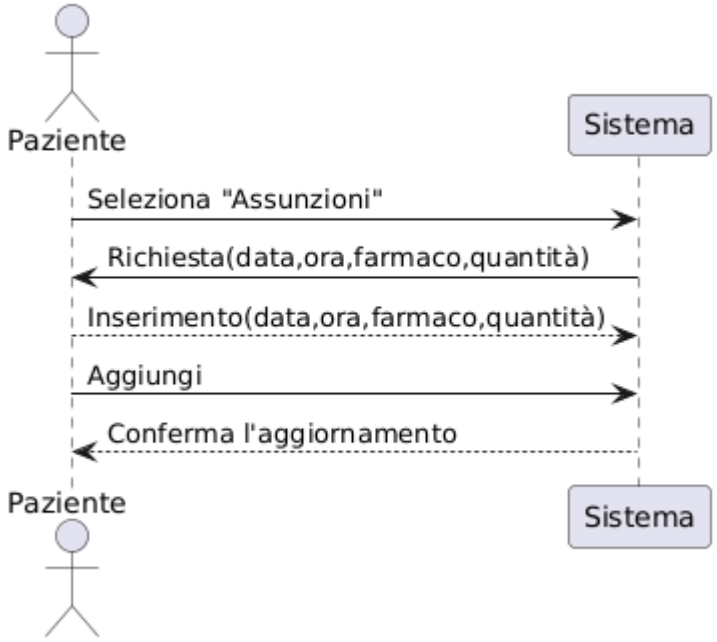

Figura 9: Sequence Diagram per **Inserire assunzioni farmaci**

Use Case: Inserire rilevazioni glicemia
Attori: paziente
Precondizioni: il paziente ha eseguito l'accesso
Sequenza degli eventi: 1. Il caso inizia dopo che il paziente clicca sulla sezione relativa alle rilevazioni 2. Il paziente immette i data, pasto e valore 3. Il paziente seleziona la funzione "Aggiungi" 4. Il sistema si aggiorna con la rilevazione
Postcondizione: la rilevazione risulta aggiunta ed è visibile anche dal diabetologo

Tabella 7: Specifiche di **Inserire rilevazioni glicemia**

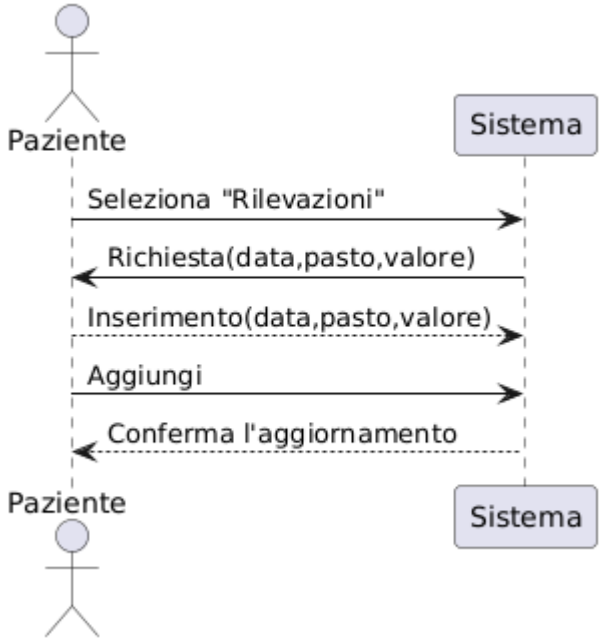

Figura 10: Sequence Diagram per **Inserire rilevazioni glicemia**

Visualizzare terapie prescritte inizia quando il paziente entra nella pagina delle assunzioni e terapie. La lista con le terapie è presente e consultabile direttamente da lì.
Inserire sintomi/patologie è opzionale e il paziente lo esegue dalla pagina delle rilevazioni, compilando e aggiungendo il sintomo/patologia da qui è affetto tramite casella di testo.
Inserire terapie concomitanti è opzionale e il paziente lo svolte dalla pagina di terapie, e proprio come per **Inserire sintomi/patologie** egli inserisce in una texbox la descrizione della terapia esterna che sta seguendo e la aggiunge, in modo che sia visibile al diabetologo per evitare combinazioni di farmaci che potrebbero essere dannose.

2.2 Diagramma di attività

NOTA: i diagrammi di attività non considerano interruzioni o ritorni per eseguire eseguire altre attività in quanto ne verrebbe meno la leggibilità. Inoltre non vengono rappresentati flussi alternativi o comunque al di fuori dei principali per lo stesso motivo.

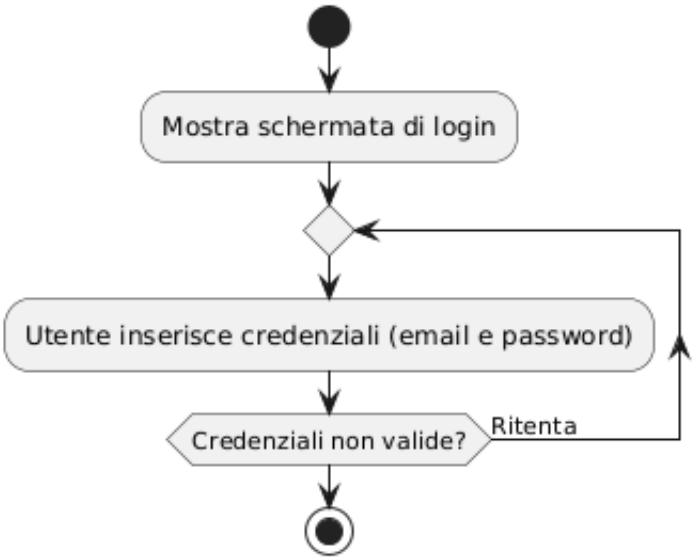

Figura 11: Activity Diagram per **Login**

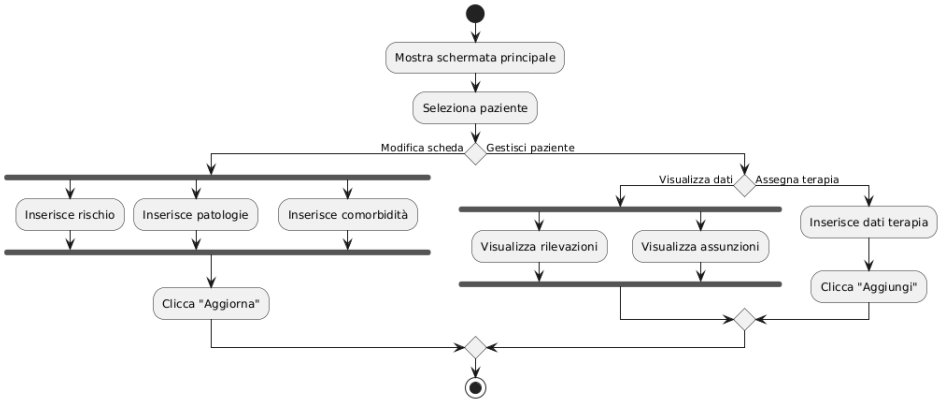

Figura 12: Activity Diagram per le **interazioni diabetologo**

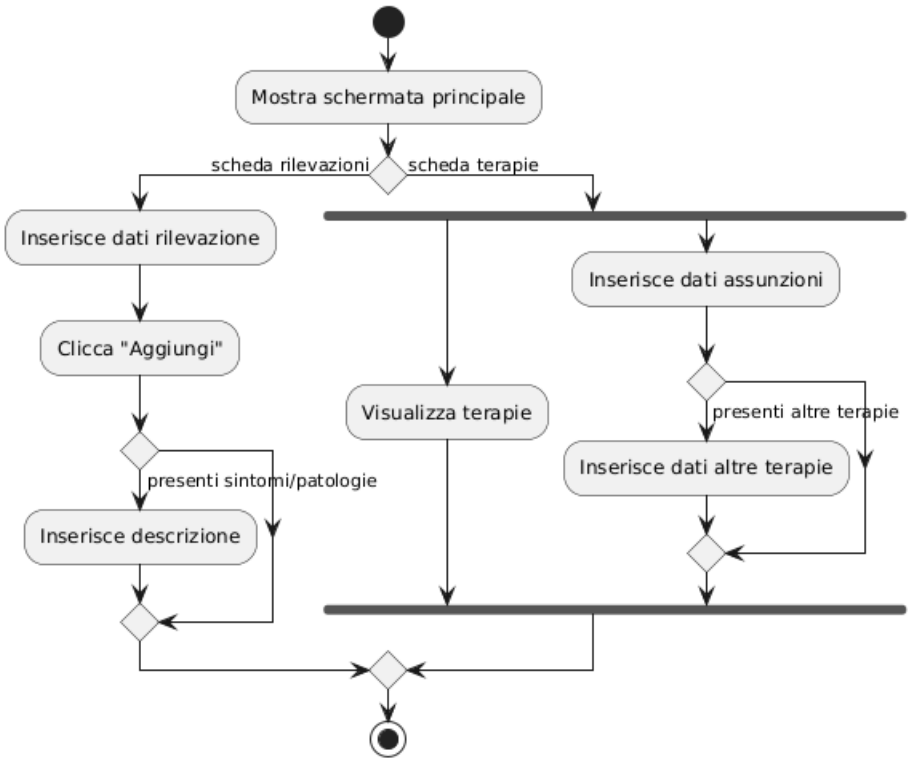

Figura 13: Sequence Diagram per le **interazioni paziente**

3 Codice

3.1 Classi utenti

Per quanto riguarda le classi principali degli utenti, questo è il Class Diagram relativo, facente parte del Class Diagram del Model.

Figura 14: Class Diagram per **utenti**

3.2 Pattern MVC

La struttura generale del progetto segue il pattern MVC. Purtroppo le dimensioni dei diagrammi di classe compresi di campi e metodi ne impediscono la visione e l'analisi, per cui seguiranno anche delle spiegazioni per ognuno di essi.

Model

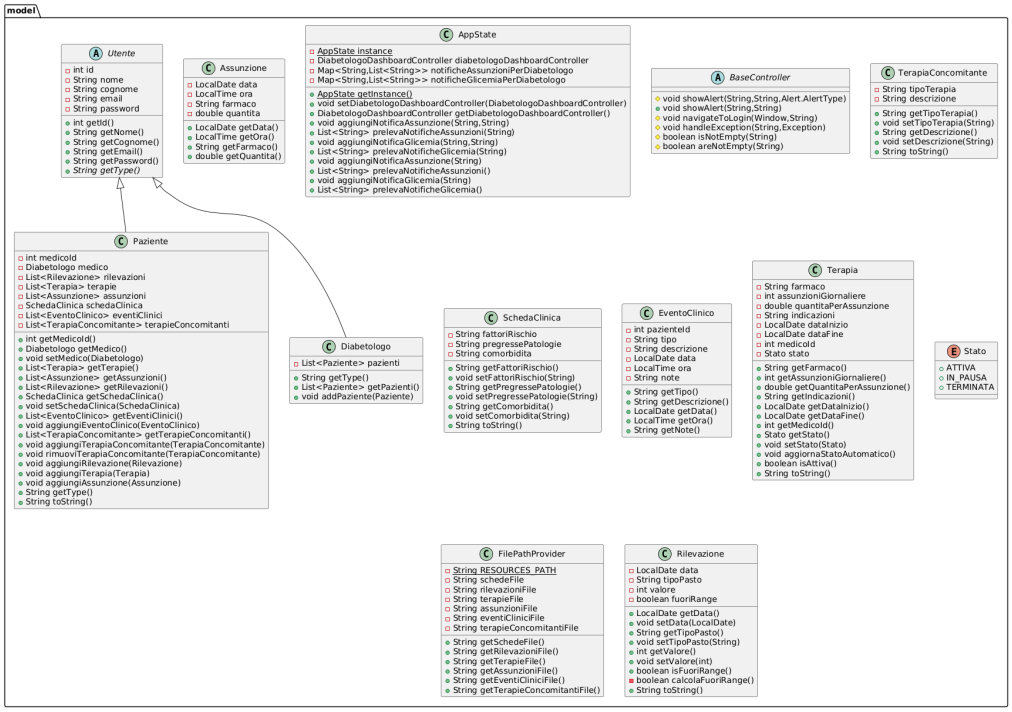

Figura 15: Class Diagram per **model**

Il model comprende anche le classi utenti **Utente** (astratta), da cui ereditano **Diabetologo** e **Paziente** (le uniche con relazione), oltre che le altre di backend che sono:

- **Assunzione** che salva i dati di una assunzione
- **Rilevazione** che salva i dati di una rilevazione

- **Stato** che è un'enumerazione per lo stato della terapia (*ATTIVA*, *IN PAUSA* e *TERMINATA*
- **Terapia** che salva i dati di una terapia
- **SchedaClinica** che salva i dati della scheda clinica di un paziente
- **TerapiaConcomitante** che salva i dati di una terapia esterna concomitante con una terapia glicemica
- **EventoClinico** per la gestione dei sintomi/patologie
- **AppState** per creare una sola istanza del programm (singleton) per permettere le notifiche
- **FilePathProvider** per la gestione dei percorsi dei file CSV e l'associazione di questi quando richiesti
- **BaseController** astratta, serve per caricare la pagina di login da qualsiasi controller e mostrare gli errori, gli alert e le validazioni di stringhe

Controller

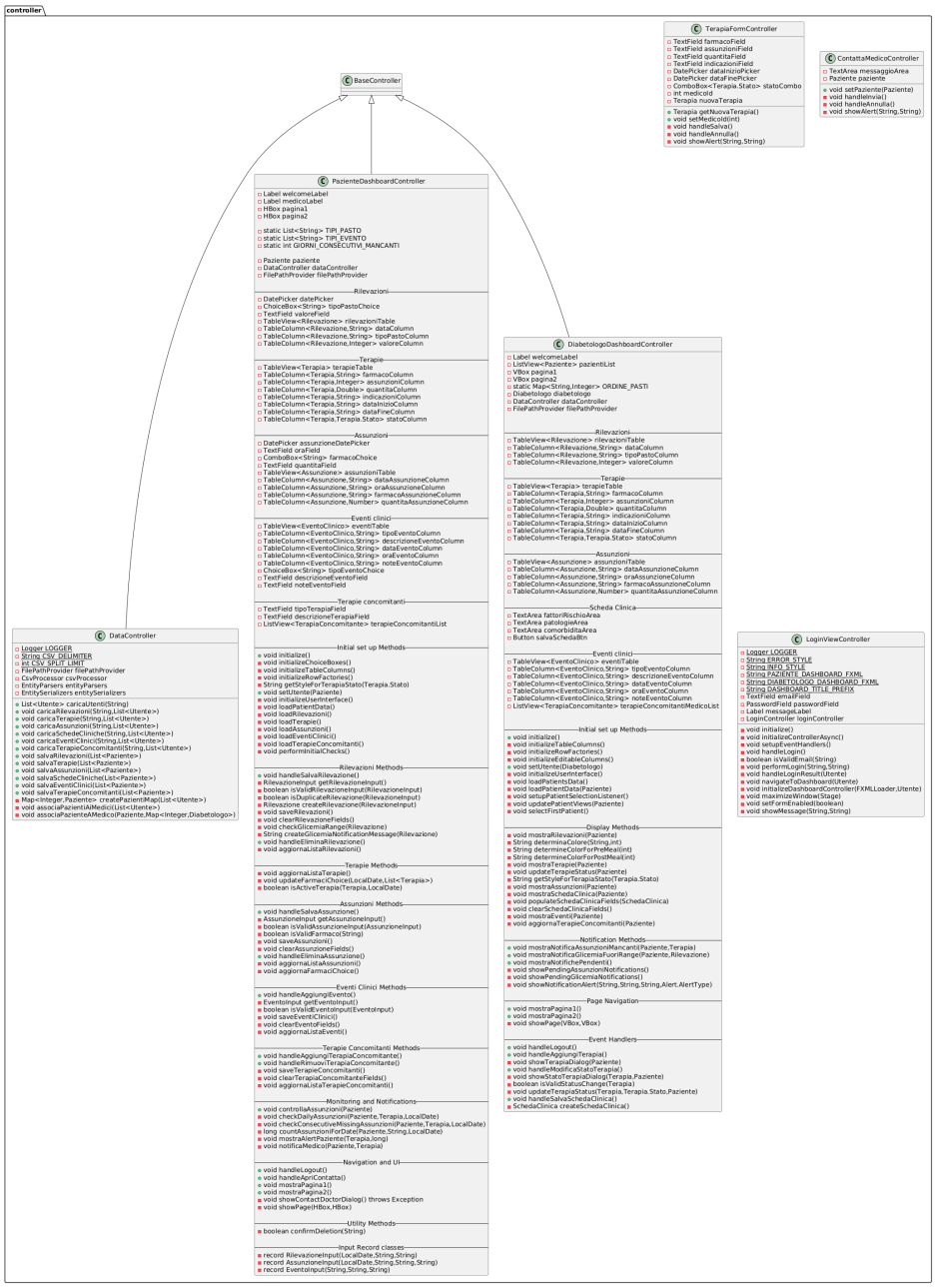

Figura 16: Class Diagram per **controller**

Il controller comprende tutte le classi per la gestione e il collegamento del backend (Model) con il frontend (View), come da pattern MVC.

- **PazienteDashboardController** (la più grande) che eredita da BaseController e gestisce tutto ciò che riguarda le pagine del paziente
- **DiabetologoDashboardController** (la seconda più grande) che come la precedente eredita da BaseController e gestisce tutto ciò che riguarda le pagine del diabetologo
- **ContattaMedicoController** per appunto gestire lo use case *Contattare medico*
- **TerapiaFormController** per gestire il form per immettere i dati delle terapie da parte del diabetologo
- **LoginViewController** per gestire la pagina di login
- **LoginController** carica gli utenti dai file CSV e poi carica i dati associati ai pazienti
- **DataController** esegue operazioni di lettura e scrittura sui file CSV

View

Il componente della View è stata creata con lo strumento *Scene Builder* che consente una costruzione in maniera grafica dell'interfaccia per l'applicazione generando automaticamente il codice JavaFX per i vari file FXML.

Riassumendo

Dove: l'intero progetto è strutturato così:

- Model: Paziente, Terapia, Assunzione, Rilevazione
- View: file FXML (TerapiaForm.fxml, LoginView.fxml, ecc.)
- Controller: LoginViewController, PazienteDashboardController, ecc.

Motivo: separa la logica di business (Model) dalla UI (View) e dalla gestione degli eventi (Controller).
Vantaggio: codice più modulare, facile da testare e manutenere
Infine è stato utilizzato lo strumento *ClaudeAI*, ossia un chatbot LLM, per eseguire il refactoring del codice rendendolo più ordinato e leggibile, per poi verificarne comunque l'elaborato e i cambiamenti apportati all'originale.

3.3 Sequence Diagram

Qui ora vengono riportate 4 interazioni di componenti del codice che sono ritenute singificative, principalmente tra i vari controller e AppData.

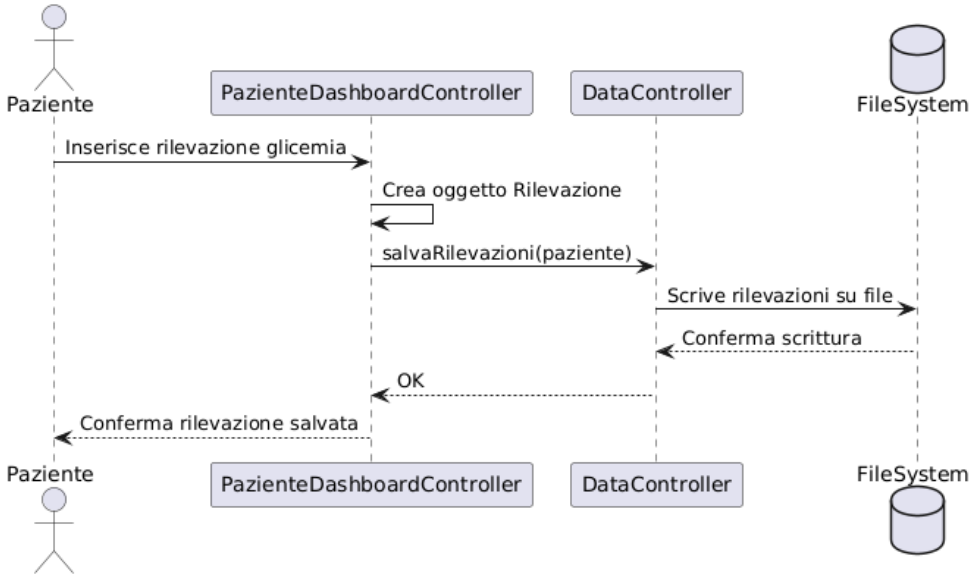

Figura 17: Sequence Diagram relativo all'aggiunta di una rilevazione da parte del paziente

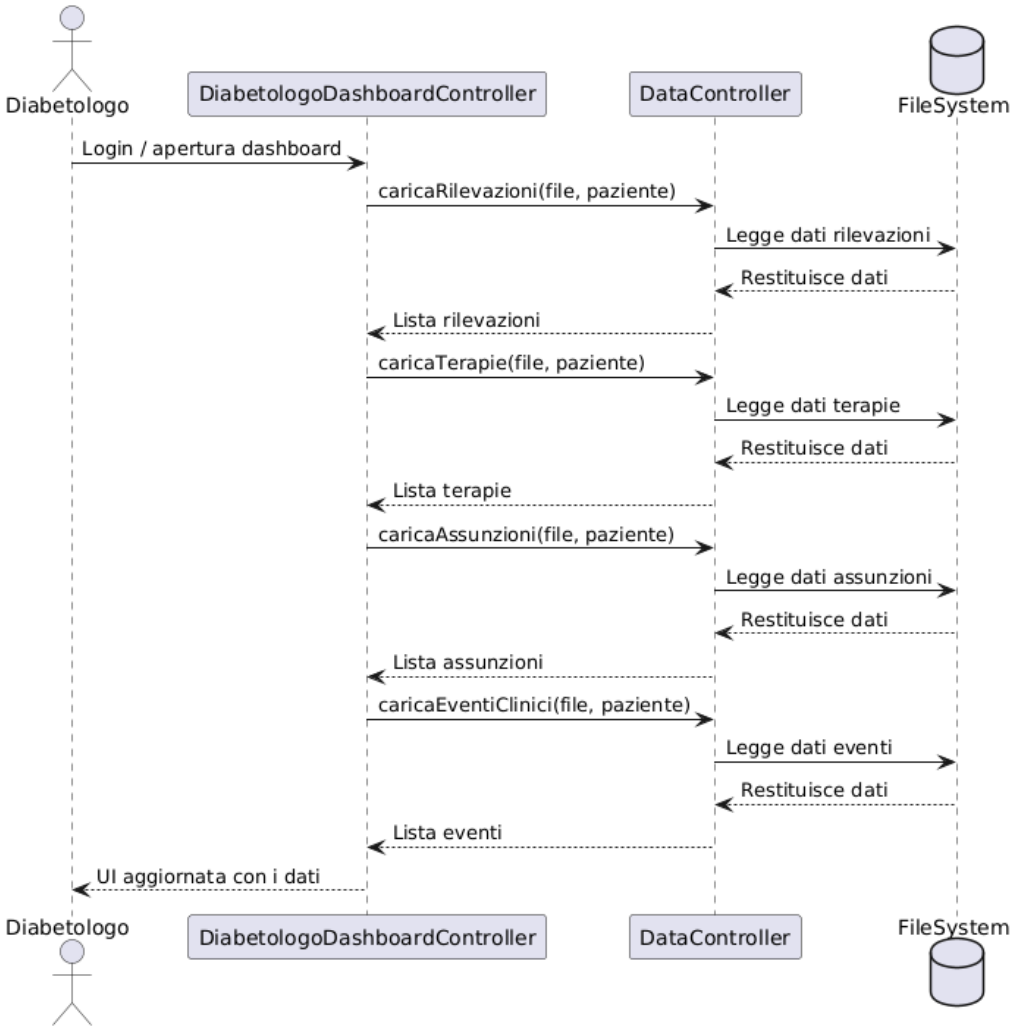

Figura 18: Sequence Diagram relativo al caricamento dei dati di un paziente per un diabetologo

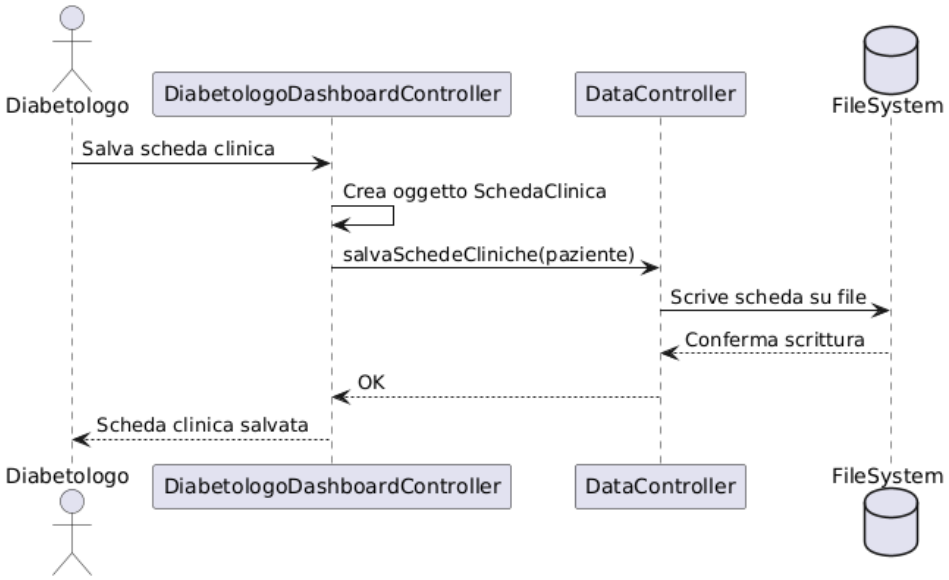

Figura 19: Sequence Diagram relativo alla modifica di una scheda clinica di un paziente da parte di un diabetologo

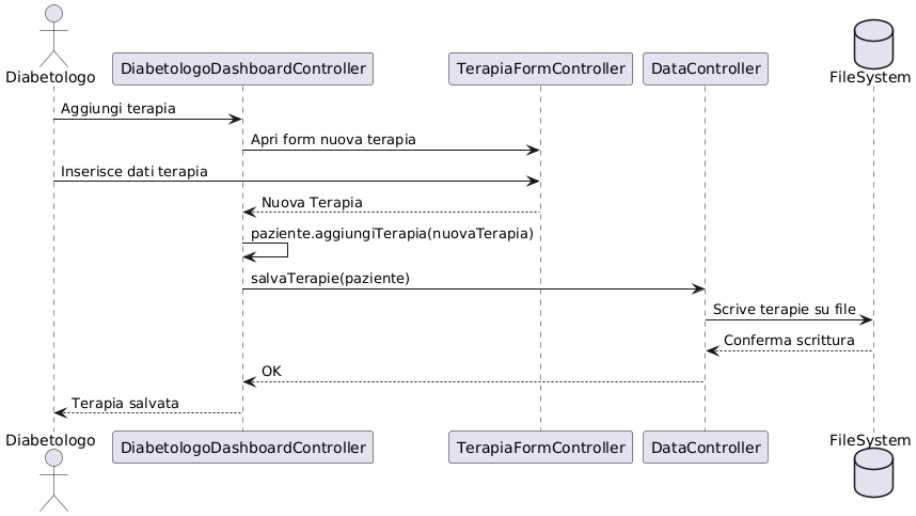

Figura 20: Sequence Diagram relativo all'aggiunta di una terapia ad un paziente da parte di un diabetologo

3.4 Pattern

Singleton (creazionale)

Dove: *AppState.java*
A cosa serve: garantisce che esista una sola istanza della classe *AppState* in tutta l'applicazione, fornendo un punto di accesso globale a questa istanza.
Utilizzo:
private static AppState instance;
public static AppState getInstance() {
if (instance == null) instance = new AppState();
return instance;
}
Viene utilizzato per mantenere lo stato globale dell'applicazione e gestire le notifiche tra i vari controller. I controller dei pazienti possono aggiungere notifiche che poi vengono recuperate dai controller dei medici.

Observer/Listener

Dove: nel sistema di notifiche tra *PazienteDashboardController* e *DiabetologoDashboardController* tramite *AppState*.
A cosa serve: definisce una dipendenza uno-a-molti tra oggetti in modo che quando un oggetto cambia stato, tutti i suoi dipendenti vengono notificati.
Utilizzo: quando un paziente registra valori fuori range o manca assunzioni, viene generata una notifica che viene poi mostrata al medico:
// Paziente aggiunge notifica
AppState.getInstance().aggiungiNotificaGlicemia(Integer.toString(paziente.getMedicoId()), msg);
// Medico recupera notifiche
AppState.getInstance().prelevaNotificheGlicemia(Integer.toString(diabetologo.getId()))

Facade (strutturale)

Dove: *DataController.java*
A cosa serve: fornisce un'interfaccia unificata e semplificata per un insieme di interfacce in un sottosistema complesso.
Utilizzo: *DataController* agisce come facade nascondendo la complessità delle operazioni di I/O, parsing CSV, e gestione dei file. I client (altri controller) usano metodi semplici come *caricaUtenti()*, *salvaRilevazioni()* e *caricaTerapie()* senza dover conoscere i dettagli di come vengono letti/scritti i file CSV.

MVC (architetturale)

Dove: nell'intera struttura del progetto.
A cosa serve: separa la logica di business (Model), la presentazione (View) e il controllo del flusso (Controller).
Utilizzo:

- Model –>Le classi base (Paziente, Diabetologo, Terapia, ecc.)
- View –>I file FXML (non mostrati ma referenziati)
- Controller –>Le classi che gestiscono l'interazione tra Model e View

Factory Method (comportamentale)

Dove: *EntityParsers.createUtente()* in *DataController.java*
A cosa serve: definisce un'interfaccia per creare oggetti, ma lascia che le sottoclassi decidano quale classe istanziare.
Utilizzo:
private Utente createUtente(String type, int id, String nome, ...) {
if (type.equalsIgnoreCase("Paziente")) {
return new Paziente(id, nome, cognome, email, password, medicoId);
} *else if (type.equalsIgnoreCase("Diabetologo"))* {
return new Diabetologo(id, nome, cognome, email, password);
} ...
}
Crea istanze di *Paziente* o *Diabetologo* basandosi sul tipo specificato nel CSV.

Template Method (comportamentale)

Dove: *BaseController.java* e le sue sottoclassi.
A cosa serve: definisce lo scheletro di un algoritmo nella classe base, permettendo alle sottoclassi di ridefinire alcuni passaggi specifici senza cambiare la struttura generale.
Utilizzo: la classe astratta *BaseController* fornisce metodi comuni come *showAlert()*, *navigateToLogin()*, *handleException()* che vengono utilizzati da tutti i controller concreti (*LoginViewController*, *PazienteDashboardController*, *DiabetologoDashboardController*). Le sottoclassi possono specializzare il comportamento mantenendo la struttura base.

Strategy (comportamentale)

Dove: *DataController.java* con le inner classes *EntityParsers* e *EntitySeria-*

lizers.
A cosa serve: permette di definire una famiglia di algoritmi, incapsularli e renderli intercambiabili. Nel progetto viene usato per gestire diverse strategie di parsing e serializzazione.
Utilizzo:
// Utilizzo di Function come strategy per il parsing
public <T>List<T>loadEntities(String filePath, Function<String, T>parser)
// Diversi parser strategies
entityParsers::parseUtente
entityParsers::parseRilevazione
entityParsers::parseTerapia
Ogni tipo di entità ha la sua strategia di parsing e serializzazione, rendendo il sistema estensibile per nuovi tipi di dati.

Riassunto

- MVC: tutta la struttura (Model/View/Controller)
- Singleton: AppState
- Observer/Listener: notifiche assunzioni
- Factory: creazione oggetti dominio
- Facade: gestione di metodi migliorata
- Template: base controller con comportamenti comuni
- Strategy: logica dei controlli delle assunzioni

4 Testing

Le funzioni sono state testate all'inizio manualmente appena implementate per risolvere problemi maggiori di sintassi e logica, oltre che qualche caso specifico. Se i test ottenevano un esito positivo si passava all'implementazione della funzione successiva e così via. I test principali riguardavano il caricamento, il salvataggio e la visualizzazione di dati dei CSV, oltre a vari altri controlli e prove per le varie funzioni.

4.1 Unit Tests

Con JUnit sono stati testati i 3 controller principali che svolgono la maggior parte del lavoro nel software e le notifiche pop up. I 3 controller sono il DataController, il PazienteDashboardController e il DiabetologoDashboardController.
NOTA: i test sono stati semplificati a causa della versione di Intellij utilizzata (quella gratuita per problemi di accesso) e a causa di essa molte librerie andavano aggiunte manualmente o spesso non funzionavano.

Test del DataController

- Test caricamento utenti da file CSV valido
- Test caricamento utenti con file vuoto
- Test caricamento utenti con righe non conformi
- Test caricamento rilevazioni
- Test caricamento terapie
- Test caricamento terapie con stato corretto
- Test caricamento assunzioni
- Test caricamento eventi clinici
- Test caricamento terapie concomitanti
- Test salvataggio rilevazioni - versione semplificata
- Test salvataggio con lista vuota

- Test createPazientiMap con reflection
- Test associazione pazienti ai diabetologi
- Test parsing utente con tipo sconosciuto

Test del PazienteDashboardController

- Test setup paziente
- Test aggiunta rilevazione
- Test aggiunta assunzione
- Test aggiunta terapia
- Test ricerca terapie per stato
- Test validazione dati rilevazione
- Test validazione dati evento
- Test ricerca duplicati rilevazione
- Test conteggio assunzioni per farmaco e data
- Test validazione farmaco esistente in terapie attive

Test per DiabetologoDashboardController

- Test setup iniziale
- Test ordinamento rilevazioni per data e tipo pasto
- Test aggiornamento automatico stato terapie
- Test validazione cambio stato terapia
- Test creazione scheda clinica
- Test ordinamento assunzioni per data e ora
- Test ordinamento eventi clinici

- Test logica notifiche assunzioni mancanti
- Test logica notifiche glicemia fuori range
- Test popolamento campi scheda clinica

Test su ControllaAssunzioni

- Controllo assunzioni complete per oggi - no alert
- Controllo assunzioni incomplete per oggi - sì alert
- Controllo assunzioni multiple incomplete - sì alert per entrambi
- Controllo senza terapie attive - nessun controllo da effettuare
- Controllo terapie terminate - nessun controllo da effettuare

4.2 Test utente

Il software è infine stato fatto testare ad utenti esterni (famigliari) e ignari delle varie funzionalità dell'applicazione per massimizzare l'effetto delle prove. Si è trattato di richiedere di eseguire determinate azioni senza spiegare come effettivamente farle, e questo anche per valutare l'intuitività dell'interfaccia e del software complessivamente. Sono stati poi raccolti i pareri e le impressioni e in base a quelle sono state effettuate modifiche principalmente all'UI e alla navigazione, così che risultassero più comode, comprensibili e meno sature.