Documentazione progetto Telemedicina

Progetto Ingegneria del Software - A.A. 2024/2025

Alessandro Mercede VR504729

Sofia De Togni VR501921

---

1. Requisiti ed interazioni utente-sistema

Il sistema "Telemedicina" supporta la gestione e il monitoraggio del diabete di tipo 2 da parte di diabetologi e pazienti. Entrambe le categorie hanno a disposizione delle credenziali (pre-fornite dagli amministratori di sistema) con cui possono effettuare l'autenticazione. Eseguito l'accesso paziente o diabetologo verranno reindirizzati alla rispettiva schermata iniziale.

Inserimento Dati Paziente
I dati iniziali del paziente come credenziali, dati personali come Nome,Cognome, Email, Codice Fiscale (utilizzato come identificativo univoco del paziente) sono già inseriti nel sistema dai responsabili del servizio, anche per questo non abbiamo ritenuto necessario creare un metodo CreatePaziente.
Per ogni paziente è specificato un diabetologo associato, questa associazione è gestita dal diabetologo stesso (vedi *Casi d'uso relativi ai Diabetologi*).

Gli utenti interagiscono con il sistema via interfaccia JavaFX:

- I pazienti possono accedere al proprio profilo, segnalare sintomi, consultare la terapia, e registrare l'assunzione dei farmaci giornalieri.

- I diabetologi possono visualizzare i pazienti associati, modificarne descrizione della patologia e terapia, visualizzare i report e analizzare i sintomi.

---

2. Specifiche casi d'uso

Casi d'uso relativi ai Pazienti

- **1. Autenticazione**
  Il paziente inserisce le proprie credenziali e accede all'interfaccia personale.

- **2. Visualizzazione Profilo e Terapia in corso**
  Il paziente accede al suo profilo personale dove visualizza la sua terapia attuale consultando nome del farmaco, durata e frequenza giornaliera prevista.

- **3. Inserimento valori glicemici**
  Il paziente inserisce i valori glicemici rilevati nella giornata, specificando se si tratta di misurazioni post-prandiali; la data e l'ora vengono registrate automaticamente dal sistema.

- **4. Compilazione diario farmacologico**
  Il paziente seleziona i farmaci e indica il numero di volte che li ha assunti durante la giornata.

- **5. Inserimento sintomi**
  Il paziente compila un modulo in cui indica uno o più sintomi provati e una breve descrizione. La data e l'ora associate ai sintomi vengono registrate automaticamente dal sistema.

- **6. Ricezione Notifiche (Alert)**
  Se il paziente non segue correttamente la terapia indicata per tre o più volte di seguito riceverà degli alert mandati dal suo diabetologo con diversi livelli di criticità.

Casi d'uso relativi ai Diabetologi

- **1. Autenticazione**
  Il diabetologo inserisce le proprie credenziali e accede all'interfaccia personale.

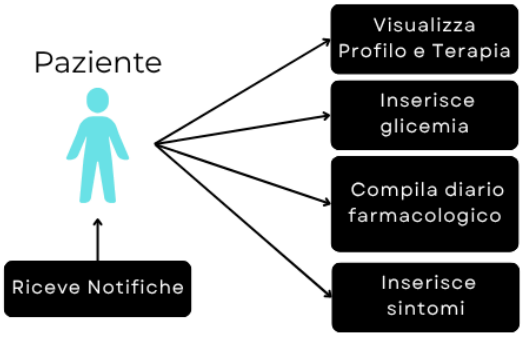
- **2. Selezione e visualizzazione pazienti associati**
  il diabetologo accede ad una tabella dove seleziona i suoi pazienti associati tramite il codice fiscale come identificativo univoco. Nella tabella saranno poi visualizzabili nome, cognome e email dei propri pazienti.

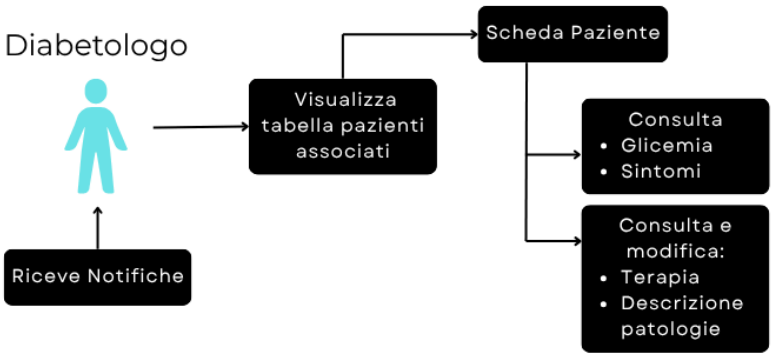
- **3. Consultazione scheda paziente**
  Il diabetologo seleziona un paziente per visualizzarne i suoi dati personali, i sintomi, le glicemie registrate e il loro andamento settimanale/mensile. Inoltre può

    - Visualizzare e inserire/modificare una descrizione riguardo informazioni importanti sul paziente (es. ex-fumatore) e/o patologie pregresse.

    - Visualizzare inserire/modificare le descrizioni relative alla terapia (nello specifico aggiungere o rimuovere farmaci).

- **4. Ricezione Notifiche (Alert)**
  Se il paziente non segue correttamente la terapia indicata il diabetologo riceverà degli alert con diversi livelli di criticità.

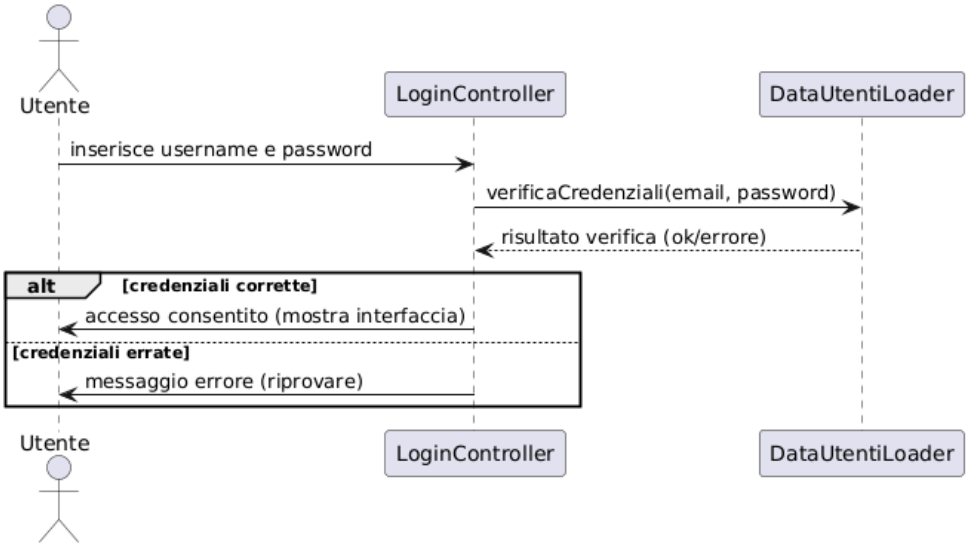




3. Diagrammi di attività

Sono stati realizzati diagrammi UML per:

- Attività di login



- paziente
- diabetologo

- (aggiungi ulm)

---

4. Sviluppo: progetto dell'architettura ed implementazione del sistema

Note sul processo di sviluppo

Inizialmente siamo partiti dalla progettazione degli schemi UML per ottenere una visione generale e strutturata dell'intero sistema. Questi schemi ci hanno permesso di individuare fin da subito le principali componenti e relazioni tra oggetti, facilitando l'organizzazione del lavoro.

In questa fase iniziale sono stati elaborati i principali **casi d'uso** e i **diagrammi di attività**. A supporto di questa fase iniziale, è stata definita anche una **prima versione dell'architettura del sistema**, con l'obiettivo di impostare una base solida e coerente per le fasi successive di implementazione.

Per quanto lo sviluppo vero e proprio, abbiamo proceduto con lo sviluppo del package **Model**, ovvero la parte logica e strutturale dei dati. Una volta definito il modello, ci siamo concentrati sull'implementazione dei **Controller**, responsabili della logica di interazione tra utente e dati. Infine, abbiamo realizzato la **View**, ovvero l'interfaccia grafica per ciascun caso d'uso, completando così il ciclo MVC.

Durante le varie fasi di sviluppo, alcuni diagrammi UML sono stati aggiornati per riflettere modifiche strutturali emerse in corso d'opera, sono stati inoltre aggiunti ulteriori UML descrittivi (Per esempio i diagrammi delle classi) . Queste modifiche hanno riguardato principalmente l'aggiunta o la rifattorizzazione di classi e metodi, sempre in funzione della coerenza tra progettazione e codice.

4.1 Architettura e pattern progettuali: Architettura MVC

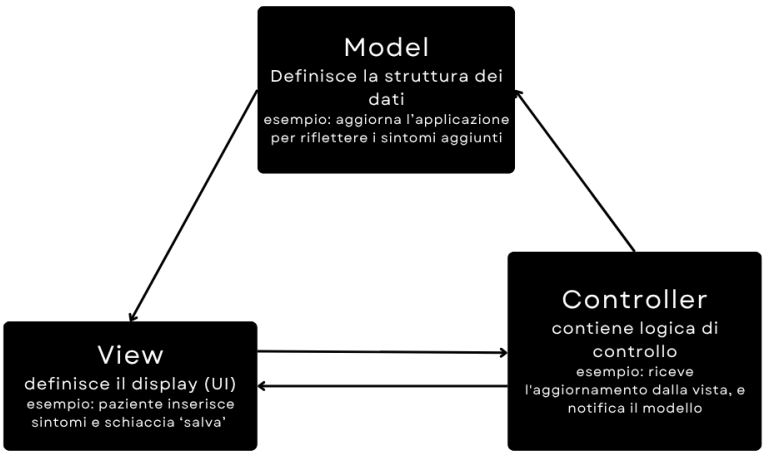
Il sistema è stato sviluppato adottando i principi della programmazione orientata agli oggetti. Dal punto di vista architetturale, si è deciso di seguire il pattern **Model-View-Controller (MVC)**, una scelta motivata dalla sua naturale integrazione con il framework **JavaFX**, impiegato per la costruzione dell'interfaccia grafica.

Questa scelta ha garantito una netta separazione tra le componenti fondamentali dell'applicazione, sia a livello logico che organizzativo, favorendo la modularità del progetto e la chiarezza nello sviluppo. Ogni componente è stata organizzata in un package dedicato:

- **Modello (Model)**: rappresenta la parte del sistema dedicata alla gestione dei dati e alla struttura delle informazioni. Include le entità fondamentali e la logica relativa alla loro memorizzazione e accesso.

- **Vista (View)**: si occupa della rappresentazione grafica del modello. I dati vengono mostrati all'utente attraverso interfacce realizzate mediante JavaFX, favorendo l'interazione in modo intuitivo e reattivo.

- **Controllore (Controller)**: definisce il comportamento del sistema in risposta alle azioni dell'utente o ad altri eventi. Collega la vista al modello, gestendo il flusso delle informazioni e le regole applicative.



4.2 Design pattern utilizzati

L'applicazione di pattern riguarda ovviamente il package Model, che gestisce l'intera mole di dati.

- Siamo partiti dall'idea di avere due attori principali, paziente e diabetologo, che con le rispettive credenziali possono autenticarsi, partendo da questa idea di base allora ci è sembrato naturale creare una classe astratta `Utente` e due classi `Paziente` e `Diabetologo` che estendono Utente, cioè abbiamo implementato il **pattern di ereditarietà**.
  La classe astratta `Utente` rappresenta un concetto generico (generalizzazione), mentre le classi `Paziente` e `Diabetologo` rappresentano varianti specializzate di quell'utente
  Questo è un principio cardine della programmazione orientata agli oggetti, noto

anche come **polimorfismo**.

- nel modello tutti costruttori sono public quindi non utilizziamo un pattern singleton, he prevede un getinstance, unico modo per avere quell'istanza.

- non è stato applicato alcun pattern Gof solo pattern della progettazione OOP

- Pur non avendo implementato formalmente il pattern X (es. Singleton, Repository...), ci siamo ispirati ai suoi principi:
  - hotpoint 1
  - hotpoint 2

  Questo ha orientato la progettazione verso [benefici].

4.3 Comunicazione tra le componenti MVC

Per la comunicazione tra Modello e Vista, così come tra Modello e Controllore, si è utilizzato il Modello come punto di accesso centralizzato alle informazioni. La gestione dei dati avviene infatti attraverso i metodi forniti dalle classi del Modello, che espongono in modo controllato le collezioni di dati e permettono alle altre componenti di recuperare o aggiornare i dati in maniera coerente.
Dettagli delle comunicazioni tra componenti:

- **Vista → Controller**
  - Tramite annotazioni `@FXML` e collegamento FXML-JavaFX.
    Funzione: la View (scritta in FXML) chiama i metodi del Controller quando l'utente interagisce con l'interfaccia (es. clic su pulsanti, inserimento dati).

Esempio pratico:
Nel file .fxml

```
 <Button fx:id="loginButton" onAction="#handleLogin"/>
```

Nel file .java

```
 @FXML
private void handleLogin(ActionEvent event) {
    // gestisce il login
}
```

- **Controller → Model**
  - Invocazione diretta **di metodi pubblici** delle classi nel package `model`.
    Funzione: il Controller elabora i dati ricevuti dalla View, li valida e aggiorna il Model (crea oggetti, modifica stati, richiede dati).

Esempio pratico:
Nel file .java

```
 Paziente paziente = new Paziente(nome, email);
model.aggiungiPaziente(paziente);
```

- **Model → Controller / View**
  - Il Controller recupera i dati dal Model e li passa alla View manualmente. Funzione: visualizzare aggiornamenti (es. nuova terapia, valori glicemia).

Esempio pratico:
Nel file .java

```
List<Glicemia> valori = model.getValoriGlicemia(paziente);
tableView.setItems(FXCollections.observableList(valori));
```

4.4 MVC: nel dettaglio

Guardiamo nello specifico cosa contengono e come operano le componenti principali del pattern MVC (Model - View - Controller)

MODEL

Il **modello** rappresenta la logica del sistema, si occupa della **struttura dei dati e della loro manipolazione**.

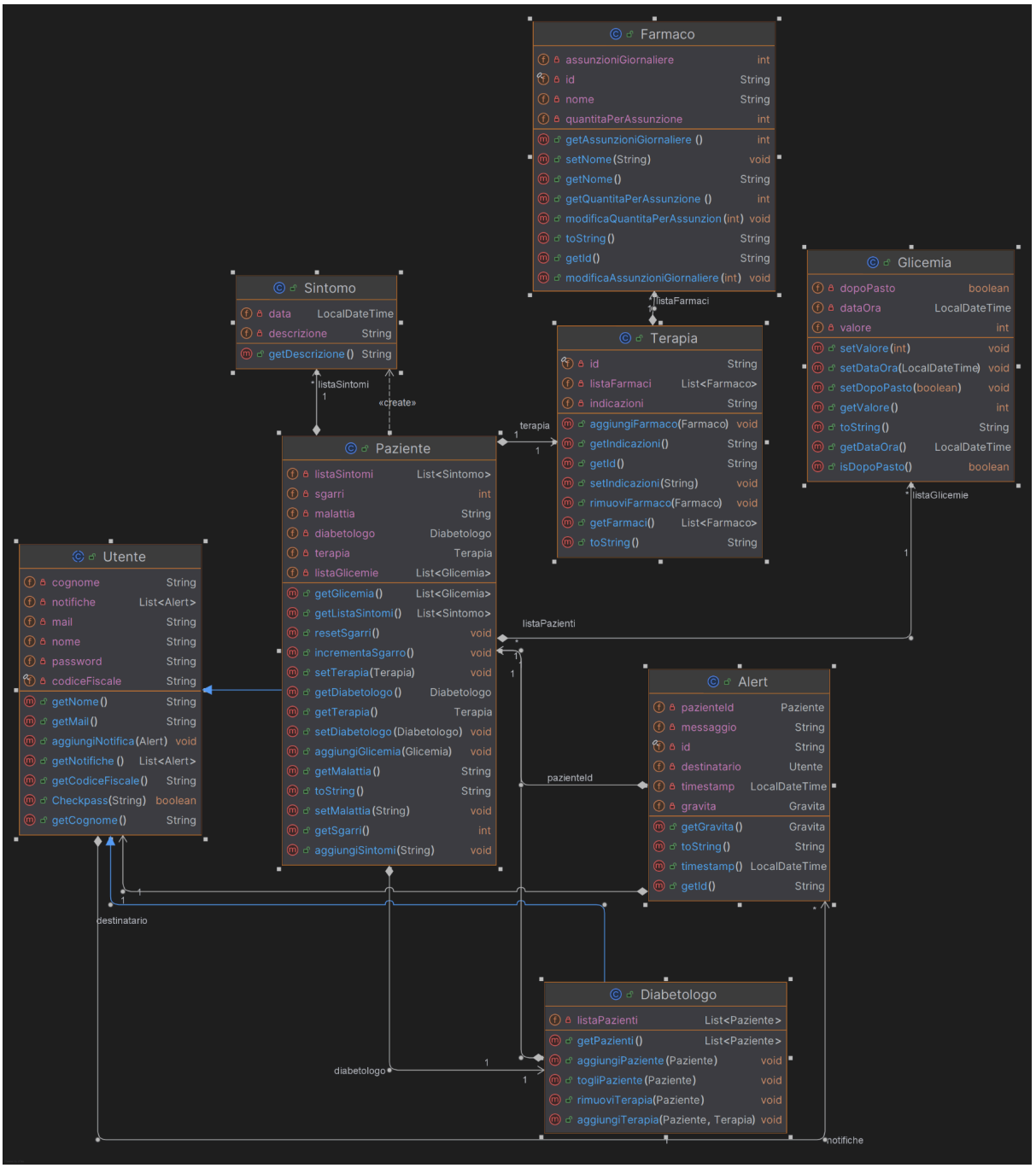
In questo progetto, include le classi che descrivono gli oggetti reali coinvolti:

- `Utente`: classe astratta che rappresenta un utente generico del sistema.
- `Paziente`: estende `Utente`, contiene informazioni personali, sintomi, rilevazioni glicemiche e terapie seguite.
- `Diabetologo`: estende `Utente`, rappresenta il medico specialista responsabile dei pazienti.
- **`Glicemia`**: rappresenta un singolo valore glicemico rilevato dal paziente (con data, valore, momento e pasto).
- Farmaco: specifica il farmaco, la quantità per assunzione e le assunzioni giornaliere.
- `Terapia` : abbiamo interpretato la terapia come una lista di farmaci e relative indicazioni.

- **`Sintomo`**: stringa descrizione dei sintomi del paziente

- **`Alert`**: classe che gestisce il tipo di alert da mandare al paziente e/o al diabetologo con la relativa criticità e timestamp.

Diagramma delle classi del Model (con campi e metodi).



VIEW

La **View** ha il compito principale di **mostrare i dati all'utente** e di **raccogliere input.** Non contiene logica applicativa o di gestione dei dati. È responsabile solo dell'**interfaccia grafica.**

- login.fxml è la prima vista con cui ci affacciamo, con le giuste credenziali grazie al LoginController entriamo nell'interfaccia del diabetologo (**`vista_diabetologo`**) o del paziente (**`vista_paziente`**).

- Per quanto riguarda il diabetologo, le viste gestite in `DiabetologoController` sono:
  - **`vista_diabetologo_ListaPazienti`**
  - **`vista_diabetologo_Paziente (selezionato)`**
  - **`vista_diabetologo_notifiche`**

- Per quanto riguarda il paziente, le viste gestite in `PazienteController` sono:
  - **`vista_paziente_profilo`**
  - **`vista_paziente_sintomi`**
  - **`vista_paziente_report`**
  - **`vista_paziente_glicemia`**
  - **`vista_paziente_notifiche`**

CONTROLLER

Il **controller** svolge un ruolo fondamentale come intermediario tra **l'interfaccia utente (vista)** e **la logica dei dati (modello)**, g**estisce la logica dell'applicazione**: riceve input dall'utente, aggiorna il modello e modifica la vista di conseguenza.

Gestione dell'autenticazione

- Il `LoginController` verifica le credenziali inserite dall'utente (paziente o diabetologo). Se l'accesso è valido, carica dinamicamente i file `.fxml` (es. vista_paziente, vista_diabetologo) in base al tipo di utente autenticato.

Controllo delle interazioni utente

- `PazienteController` in generale gestisce l'intera interfaccia del paziente. Quando un paziente inserisce una nuova glicemia o un sintomo, il controller raccoglie questi dati e li passa al modello.
  Gestisce la terapia assegnata e i farmaci assunti dal paziente (controllando anche se sono corretti).

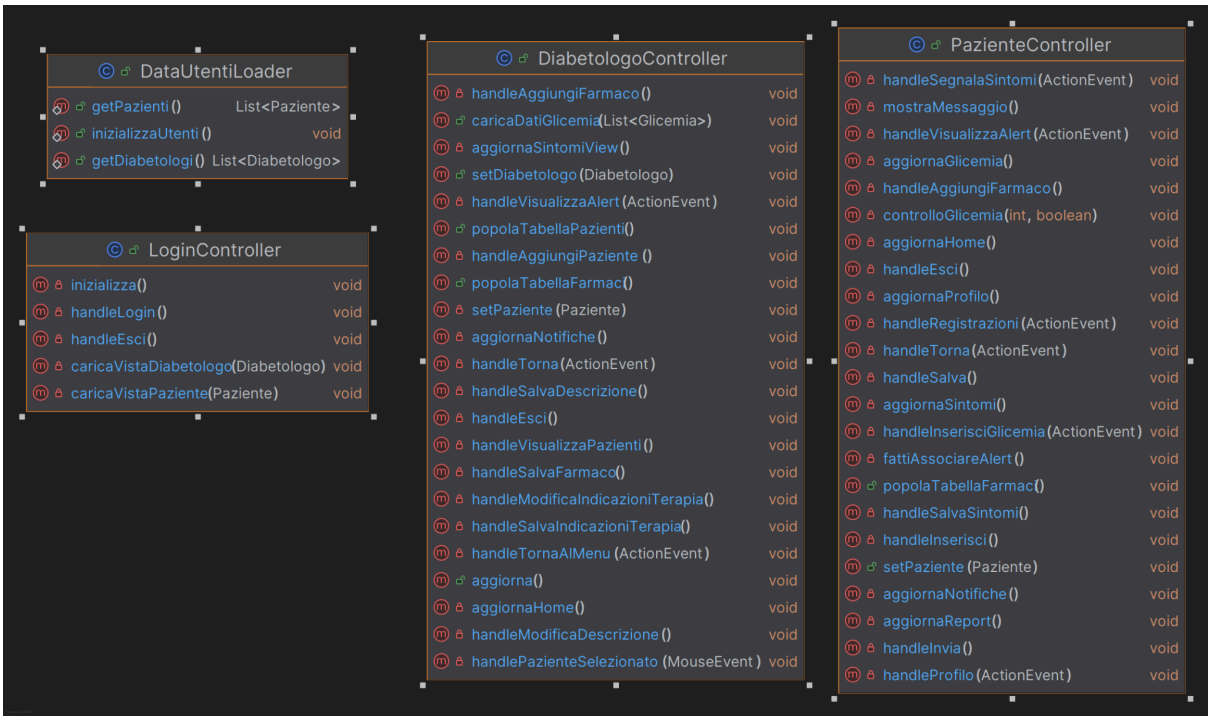
- `DiabetologoController` invece gestisce la vista dell'interfaccia del diabetologo. Viene strutturata la tabella dei pazienti associati e dello specifico paziente

selezionato. In quest'ultima può consultare i dati del paziente, i suoi valori glicemici, la sua terapia, altre patologie, essenzialmente tutto ciò descritto nel caso d'uso relativo al diabetologo **"**Consultazione scheda paziente**"** e visualizzare gli alert incluso.

Inizializzazione dei dati

- `DataUtentiLoader` inizializza alcune informazioni iniziali per far funzionare correttamente il progetto, sono quelle informazioni che vengono inserite dai responsabili del sistema come le credenziali e i dati personali del paziente. Abbiamo ritenuto opportuno inizializzare anche una terapia, alcuni sintomi e i valori glicemici di un paziente per mostrare il funzionamento del grafico e della tabella.

Diagramma delle classi del Controller, con metodi.



---

5. Test e validazione

Per assicurarsi la solidità e il corretto funzionamento del software (e corretta documentazione) abbiamo eseguito diversi controlli:

- Controllo diagrammi uml e conformità con il codice
- Ispezione del codice, verifica della correttezza dei pattern
- Test degli sviluppatori sul software

5.1 Test condotto dagli sviluppatori

Durante questa fase, abbiamo eseguito una serie di test manuali immettendo nel sistema diversi tipi di input, sia corretti che errati, per verificare che il comportamento del software fosse conforme alle aspettative. I principali test effettuati sono stati:

- **Autenticazione**: verifica che l'accesso con credenziali errate venga correttamente respinto e che, in caso di credenziali valide, l'utente venga indirizzato alla schermata iniziale relativa al proprio ruolo.

- **Gestione sintomi**: tentativo di salvare una stringa vuota relativa ai sintomi; il sistema ha gestito correttamente la situazione senza errori.

- **Aggiunta farmaco non valido**: inserimento di una terapia con farmaco “vuoto”; il sistema chiede al diabetologo di specificare un farmaco valido.

- **Inserimento segnalazione**: verifica che una segnalazione venga visualizzata esclusivamente dal diabetologo associato al paziente, e non da altri.

- **Accesso pazienti**: conferma che ogni medico abbia accesso solo ai pazienti di propria competenza.

- **Input non validi**: test di robustezza con inserimento di dati errati, vuoti, malformati o numeri negativi.

- **Associazione automatica farmaco-terapia**: verifica del corretto funzionamento del sistema nella selezione automatica dei farmaci pertinenti in presenza di più terapie assegnate al paziente.

5.2 Test utente generico

Come fase conclusiva, il software è stato testato da alcuni utenti con scarsa familiarità con strumenti informatici e nessun coinvolgimento nello sviluppo del progetto. L’obiettivo principale era osservare l’utilizzo del sistema in modo spontaneo, senza fornire indicazioni. Agli utenti è stata fornita solo una breve descrizione degli scopi generali del software. Successivamente, sono stati lasciati liberi di esplorare le funzionalità in autonomia.

Questo approccio ha permesso di identificare problemi non emersi durante i test interni, legati all’intuitività e all’usabilità dell’interfaccia. In generale non sono stati riscontrati problemi strutturali e il test è stato svolto senza riscontrare gravi problematiche.