TITOLO RELAZIONE

### A cura di

## INDICE:

## Requisiti ed interazioni utente-sistema 2

## Specifiche casi d’uso . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2

## Casi d’uso relativi ai Medici . . . . . . . . . . . . . . . . . . . . . 3

## Casi d’uso relativi ai Farmacologi . . . . . . . . . . . . . . . . . . 6

## Diagrammi di attività . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

## Sviluppo: progetto dell’architettura ed implementazione del sistema 13

## Note sul processo di sviluppo . . . . . . . . . . . . . . . . . . . . . . . 13

## Progettazione e pattern architetturali usati . . . . . . . . . . . . . . . 14

## Breve commento sulle interfacce . . . . . . . . . . . . . . . . . . . 18

## Implementazione e design pattern usati . . . . . . . . . . . . . . . . . 19

## Generalità . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19

## Classe Model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20

## Diagrammi di sequenza del software implementato . . . . . . . . 20

## Attività di test e validazione 27

## Ispezione codice e documentazione . . . . . . . . . . . . . . . . . . . . 27

## Unit test . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27

## Test degli sviluppatori . . . . . . . . . . . . . . . . . . . . . . . . . . . 29

## Test utente generico . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30

## Appendice: creazione di oggetti mediante la classe Model 31Requisiti ed interazioni utente-sistema

## Specifiche casi d’uso

## Il sistema proposto supporta la gestione per la prenotazione di passaporti da parte del personale addetto e le richieste di prenotazione per il cittadino, inizialmente identificati come utente generico. Solo successivamente, in base al percorso scelto vengono differenzializzati. L’applicazione Il cittadino è diviso in registrato(possibilità di fare login) e non registrato (registrazione necessaria).

## Nel primo caso, effettuato il login, il cittadino verrà indirizzato alla pagina iniziale che gli permetterà di fare richiesta per rinnovo/smarrimento/nuovo passaporto ecc…, in base alle disponibilità delle sedi gestite dal personale. Nel secondo caso è necessario che il cittadino si registri nel sistema altrimenti non può fare richiesta.

Il personale ha delle credenziali( pre-fornite dagli amministratori del sistema)con cui

## possono effettuare l’autenticazione. In caso questa vada a buon fine, il personale

## verrà reindirizzato alla rispettiva schermata iniziale.

## Casi d’uso relativi a utente generico

L’utente si interfaccia con il sistema, dove viene subito invitato a scegliere il suo ruolo con conseguente autenticazione o registrazione (se non è registrato).

Nella home sono presenti due pulsanti: “Accesso personale amministrativo” e “Accesso per il cittadino”.

c

**Scelta “accesso personale amministrativo”**

L’utente generico che si identifica come personale amministrativo sarà proprietario di

un utente ed una password (pre-forniti dall’amministrazione) per poter accedere al sistema amministrativo.

**Scelta “accesso per il cittadino”**

L’utente generico che si identifica come cittadino, se privo di registrazione, deve necessariamente registrarsi nel sistema affiche quest’ultimo possa verificare la sua identità.

L’utente già registrato, può effettuare il login per entrare nel sistema.

|  |
| --- |
| *Attori: Utente generico**Precondizioni: Nessuna**Passi:**1. L’utente si interfaccia al sistema**2. L’utente* *(a). si identifica come “personale amministrativo” ed accede al sistema di amministrazione*  *(b). si identifica come “cittadino” ed accede al sistema di prenotazione* *Postcondizioni: l’utente generico diventa specifico.* |

## Casi d’uso relativi al personale di amministrazione

Dopo opportuna autenticazione tramite la pagina dedicata, il personale d’A. viene introdotto nella home dell’amministrazione che permette di inserire il numero di “slot appuntamenti” per le varie fasce orarie nelle varie sedi.

**Inserire gli “slot disponibilità”**

Il personale d’A. deve poter inserire il numero di disponibilità per gli appuntamenti. Per fare questo, è necessario scegliere la sede,il giorno, la relativa fascia oraria e il numero di posti disponibili.

|  |
| --- |
| *Attori: Personale Amministrazione**Precondizioni: Il personale deve essersi autenticato.**Passi:**1. Il personale accede al sistema**2. Il personale è introdotto all’interfaccia di base* *3. Il personale sceglie il giorno da “modificare”* *4. Il personale inserisce gli “slot appuntamenti” per la sede scelta tra le disponibili.**5. Il personale conferma il numero di slot per quella fascia oraria giornaliera in quella sede* *Postcondizioni: la disponibilità è inserita.* |

## Casi d’uso relativi al cittadino

**Note generali:**

Il cittadino per poter accedere al sistema deve essere registrato.

Se non è registrato deve effettuare necessariamente la registrazione ( per verificare l’identità del cittadino tramite CODICE FISCALE).

Dopo opportuna autenticazione tramite la pagina dedicata, il cittadino viene introdotto nella home nella quale può scegliere tra i pulsanti “richiesta passaporto” o “ritiro passaporto” per cui sucessivamente potrà scegliere uno “slot disponibilità” tra quelli resi disponibili dall’amministrazione. Presenti anche due finestra nella quale il cittadino può vedere i suoi prossimi appuntamenti e i documenti necessari per quest’ultimi. Ha la possibilità una volta finite le azioni, di eseguire il logout.

**Richiesta passaporto:**

Nell’eventualità in cui il cittadino prema il pulsate “richiesta passaporto”, il cittadino verrà introdotto in una schermata in cui deve selezionare il motivo della richiesta tra:

- Rilascio passaporto per la prima volta

- Furto

- Rilascio passaporto per scadenza del precedente

- Smarrimento

- Deterioramento

Successivamente verrà introdotto nella pagina delle disponibiltà in cui gli sarà presentato un calendario del mese corrente con il numero degli slot disponibili delle varie fasce orarie nei giorni resi disponibili dal personale amministrativo nella sede scelta; il quale dovrà scegliere per prendere appuntamento.

**Ritiro passaporto:**

Nell’eventualità in cui il cittadino prema il pulsate “richiesta passaporto”, il cittadino verrà introdotto in una schermata similare alla precedente per la richiesta in cui gli sarà reso disponibile un calendario del mese corrente con il numero degli slot disponibili delle varie fasce orarie nei giorni resi disponibili dal personale amministrativo; il quale dovrà scegliere per prendere appuntamento.

|  |
| --- |
| *Attori: Cittadino**Precondizioni: Il cittadino deve registrarsi**Passi:**1. Il cittadino accede al sistema dedicato**2. Il cittadino è introdotto all’interfaccia di base* *3. Il cittadino*  *(a).se è la prima volta, si registra*  *(b).se è gia registrato, effettua l’autenticazione* *4. Il cittadino sceglie la sede dove vuole prendere appuntamento**5. Il cittadino sceglie il giorno tra quelli disponibili per l’appuntamento**6. Il cittadino sceglie la fascia oraria del giorno scelto**7. Il cittadino conferma l’appuntamento**Postcondizioni: l’appuntamento è stato inserito.* |
|  |

## Diagrammi di attivita:

***Nota:*** *i seguenti diagrammi catturano una singola attività di un utente rispetto al sistema. Non è stata rappresentata nel diagramma la possibilità di ripetere più volte la stessa operazione, in sequenza, senza chiudere il software. Questo per semplici ragioni di chiarezza e leggerezza; si considerano quindi le singole attività d’interazione*

**Sviluppo: progetto dell’architettura ed implementazione del sistema**

**Note sul processo di sviluppo**

Il processo di sviluppo è stato essenzialmente di tipo Agile ed Incrementale. Tuttavia, si è cercato quanto più possibile di mantenere sequenziali (seppure inserite all’interno di un ciclo) le fasi di progettazione, implementazione e validazione. Questo è stato fatto semplicemente per cercare di procedere sempre prima progettando e dopo implementando, con l’obiettivo di avere sempre un terreno solido e pensato sul quale lavorare. Dopo ogni modifica significativa (versione) è stata condotta una breve attività di test. Si noti che queste fasi non sono sempre state lineari e propellenti, ma hanno anche incluso attività di refactoring sul materiale già costruito. Durante ogni ciclo, e parallelamente a queste tre attività, si è condotta ciò che possiamo dire genericamente documentazione. In questa attività, si è raccolto il materiale UML generato dalle fasi di test e di progettazione (principalmente diagrammi di classe) nel presente documento complessivo, aggiungendo inoltre ulteriori UML descrittivi (i sequence diagram, ad esempio, sono stati prodotti dopo aver implementato i metodi relativi). Prima di cominciare il ciclo principale, si è condotta la fase di analisi dei requisiti, generando i relativi use-cases e i diagrammi di attività. Anche la (semplice) progettazione architetturale è stata fatta prima di iniziare il lavoro centrale, in modo da assicurarsi di trovarsi per lo meno in una situazione solida da quel punto di vista. Per quanto riguarda l’implementazione, non sono state fatte grosse divisoni o piani di sviluppo programmatici. Si è seguito l’ordine prioritario di sviluppo andando ad aggiungere ciò che era ritenuto necessario man mano.

Progettazione e pattern architetturali usati

Il sistema è stato implementato basandosi su tecniche per la modellazione/programmazione ad oggetti.

Dal punto di vista architetturale, si è scelto di utilizzare il pattern MVC.

Le ragioni sono semplici: si è pensato che, essendo nativamente implementato

tramite le librerie Swing (utilizzate per produrre l’interfaccia grafica del progetto),

fosse la scelta più comoda e solida.

JavaFX è una famiglia di software applicativi, basati sulla piattaforma Java, per la creazione di rich Internet application, applicazioni web che hanno tutte le caratteristiche e funzionalità delle comuni applicazioni per computer

Ciò ha permesso di separare nettamente e comodamente le tre componenti a

livello logico e a livello di sviluppo. Ogni membro è stato inserito in un package

separato.

• Modello: sottoparte del sistema che riguarda i dati e le informazioni

memorizzate. Definisce la struttura dell’informazione gestita dal sistema.

• Vista: sottoparte del sistema che rappresenta visivamente il modello, e

quindi, i dati del sistema. Si è utilizzato il framework Swing per realizzare

questa parte.

• Controllore: sottoparte del sistema che definisce la logica applicativa,

ovvero il comportamento del sistema a fronte degli stimoli esterni. Consiste

nei listener che ascoltano la vista.

Le azioni dell’utente saranno catturate dai listener. I listener saranno istruiti

a reagire di conseguenza, andando a modificare le informazioni contenute nel

Modello e quindi ad aggiornare la vista, che rappresenta il modello.

Non sono state operate ulteriori articolazioni: modello, vista e controllore sono

tre blocchi monolitici. Questo perché si è considerato che i vari elementi costitutivi giacessero sullo stesso piano.

Seguono un semplice schema dell’architettura del sistema ed i diagrammi UML

delle classi del Modello e della Vista-Controllore.

Le interazioni salienti tra Modello-Vista-Controllore saranno visibili nei diagrammi di sequenza (cfr. Implementazione).

Breve commento sulle interfacce

Per gestire la comunicazione tra Modello, Vista e Controllore, sono state adottate

le seguenti scelte.

Per la comunicazione Vista-Controllore, si è banalmente utilizzato il meccanismo

basato su ActionListeners. L’implementazione è standard, per cui possiamo

sorvolare.

Per la comunicazione Modello-Vista e Modello-Controllore, si è utilizzato come

punto di accesso alle informazioni la classe Model, che immagazzina e fornisce

i dati. Per i dettagli riguardo la sua implementazione si rimanda alla sezione

successiva; segue un breve riassunto delle comunicazioni Modello-Vista e ModelloControllore per quanto concerne l’accesso alle collezioni di dati, che non sono

altro che i metodi esportati dalla classe Model. Per quanto riguarda l’invio dei

messaggi ai singoli oggetti (una volta ottenuti dal Model) non è stato posto alcun

vincolo particolare.

• getInstace(): per ottenere l’oggetto univoco Model

• getAutenticabili(): ottieni la lista di aventi credenziali

• getFarmaci()

• getReazioniAvverse()

• getPazienti()

• getSegnalazioni()

• aggiornaTerapie(): aggiungi la terapia voluta al paziente

• getAvvisi()

• generaAvviso(): invia/costruisce un nuovo avviso

• generateIfWasWeekend(): genera un avviso weekend se è il caso

• getUnread(): gli avvisi non letti

• getFarmacoByName(): ottieni il farmaco relativo al nome passato

• createPaziente(): costruisce un nuovo paziente

• createReazione(): costruisce una nuova reazione avversa

• makeSegnalazione(): invia/costruisce una nuova segnalazione

• load(): carica i dati dal disco

• save(): salva i dati su disco

Attività di test e validazione

Per verificare la solidità del software prodotto, si sono svolte le seguenti attività:

1. Ricognizione del documento delle specifiche e confronto con i diagrammi

prodotti

2. Verifica della consistenza tra diagrammi e codice prodotto

3. Ispezione del codice, verifica della correttezza dei pattern, ricerca di

malepratiche varie

4. Unit test automatizzato mediante JUnit per verificare la correttezza di

alcune delle classi più problematiche

5. Test degli sviluppatori sul software

6. Test utente generico sul software

Ispezione codice e documentazione

In questa fase si è semplicemente rivisto il documento delle specifiche e lo si è

confrontato con i diagrammi UML prodotti, per verificare la correttezza degli use

case, activity diagrams e diagramma delle classi. Una volta finita questa attività

si è confrontato il codice (staticamente) ai diagrammi UML, per verificarne la

consistenza. Infine si è data una nuova ispezione del codice per cercare infrazioni,

malepratiche e cattivi usi dei pattern. Particolare attenzione è stata data al non

aver creato autonomamente oggetti delegati alla classe Model (factory pattern).

Unit test

In questa fase è stata creata una versione alternativa del codice ai fini del test

automatizzato con JUnit. Si sono testate alcune delle classi più problematiche e

alcuni delle funzionalità più complesse.

import java.util.HashMap;

import Modello.Model;

import Modello.Segnalazione;

public class TestFarmaco extends Segnalabile implements java.io.Serializable{

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

private String name;

boolean ritirato = false, controllato = false, monitorato = false;

private HashMap<Integer, Integer> worryingCasesPerYear = new HashMap<>();

27

public TestFarmaco(String name) {

this.name = name;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public void setRitirato() {

ritirato = true;

}

public void setControllato() {

controllato = true;

}

public void setMonitorato() {

monitorato = true;

}

public int hashCode() {

return name.hashCode();

}

public String toString() {

return name + getStatusString();

}

public String getStatusString() {

if(ritirato)

if(monitorato)

if(controllato)

return " [R, M, C]";

else

return " [R, M]";

else

if(controllato)

return " [R, C]";

else

return " [R]";

else

28

if(monitorato)

if(controllato)

return " [M, C]";

else

return " [M]";

else

if(controllato)

return " [C]";

else

return "";

}

public void addSegnalazione(Segnalazione seg) {

super.addSegnalazione(seg);

if(seg.getReazioneAvversa().getLivelloGravita() > 3) {

worryingCasesPerYear.putIfAbsent(seg.getDataReazioneAvversa().getYear(), 0); // se era nullworryingCasesPerYear.put(seg.getDataReazioneAvversa().getYear(), //aumento il counter di 1

worryingCasesPerYear.get(seg.getDataReazioneAvversa().getYear()) + 1

);

if( worryingCasesPerYear.get(seg.getDataReazioneAvversa().getYear()) % 10 == 0 &&

worryingCasesPerYear.get(seg.getDataReazioneAvversa().getYear()) > 0) //

Model.getInstance().generaAvviso("Il TestFarmaco " + this + " ha ricevuto " + worryingCasesP" casi gravi nel " + seg.getDataReazioneAvversa().getYear() + ".");

}

}

public boolean equals(Object o) {

return o instanceof TestFarmaco && ((TestFarmaco)o).getName().equalsIgnoreCase(name);

}

}

Test degli sviluppatori

In questa fase lo sviluppatore ha immesso nel sistema degli input (sia corretti

sia errati) per vedere se la reazione del software fosse quella attesa. Alcuni dei

test svolti sono:

• Verifica del corretto funzionamento dell’autenticazione: i dati errati vengono respinti, ed a fronte dei dati corretti all’utente viene mostrata

la schermata iniziale legata alla sua professione.

• Farmacologo a database vuoto: si è provato a pigiare ogni pulsante

pigiabile in assenza assoluta di dati, per vedere se questo creasse eccezioni

od errori.

• Medico a database vuoto: si è provato a pigiare ogni pulsante pigiabile

29

in assenza assoluta di dati, per vedere se questo creasse eccezioni od errori.

• Inserimento di una segnalazione. Verifica che tale segnalazione sia

poi visibile dai farmacologi e dal medico, ma non da altri medici.

• Verificato che il medico veda solo i suoi pazienti ma tutte le reazioni

in sistema

• Verificato il sistema di avvisi (settimanale, soglia dei 50, soglia dei 10)

• Verifica di vari inserimenti di input errati, input vuoti, stringhe malformate, numeri negativi

• Verifica della reazione del software all’inserimento di terapie finite prima di

iniziare, o di segnalazioni senza alcuna terapia, terapie prima della nascita

di un paziente, o di reazioni non ancora avvenute (controllo sulle date)

• Inserimento di reazioni, farmaci o segnalazioni duplicate

• Controllato che il meccanismo di segnalazione inferisca i farmaci corretti se vi sono più terapie a carico del paziente (ma solo alcune attive

durante la reazione avversa)

Test utente generico

Come ultimo scaglione il software è stato sottoposto ad un test da parte di

alcuni individui con limitata dimestichezza informatica ed assoluto distacco

dallo sviluppo. In questa fase non si è cercato in nessun modo di guidare o

strutturare l’esperienza, per non influenzare in alcun modo il risultato; piuttosto

si è lasciato che il soggetto navigasse liberamente il sistema. Non è stata data

nessuna spiegazione sull’utilizzo del software, se non una generale indicazione

dei suoi fini; ci si è limitati a rispondere alle domande, quando sollevate. L’unico

scopo del test era quello di rilevare errori invisibili allo sviluppatore; in realtà,

gli utenti a cui è stato mostrato il software hanno anche aiutato ad individuare

nuove funzionalità per migliorare l’usabilità generale del sistema.

30

Appendice: creazione di oggetti mediante la

classe Model

La classe Model riprende la filosofia alla base del factory pattern: infatti, funge

da punto di creazione per varie classi delicate.

Nonostante sia stato tralasciato l’utilizzo di un’interfaccia comune, si è adottata

l’idea di delegare la creazione degli oggetti ad un unico responsabile. La classe

Model, infatti, crea oggetti di varie classi, come Paziente, ReazioneAvversa,

Segnalazione; questo per il semplice motivo che la creazione di questi oggetti è

molto delicata, ed è quindi meglio centralizzarla e delegarla ad un unico ente.

La scelta di non usare l’interfaccia comune deriva dal fatto che, sostanzialmente,

gli oggetti creati non sono logicamente assimilabili in unica interfaccia e vengono manipolati separatamente dal resto del software. Utilizzare un’interfaccia

comune avrebbe aggiunto gradi di complicazione inutili. Chiaramente, potrebbe

sembrare che il lasciarsi indietro l’utilizzo dell’interfaccia comune sconfigga il

senso stesso dell’ultizzo di questo pattern: a questo punto, in un qualsiasi punto

del codice si potrebbe creare una classe tra quelle sopracitate in malo-modo.

La cosa è stata gestita in tre modi: innanzitutto, si è reso i costruttori degli

oggetti interessati package-protected. Inoltre, si richiede come parametro del

costruttore l’oggetto creante (this): se non è istanza di Model, viene generata

un’eccezione FactoryException. Un ulteriore meccanismo di sicurezza in tal

senso è fornito dal fatto che non è possibile aggiungere dall’esterno oggetti allo

stato di Model.

Di seguito la lista di oggetti sottoposti alla creazione tramite Model.

• Autenticabile e sottoclassi

• Paziente

• Segnalazione

• Farmaco

• Avviso

• ReazioneAvversaElenco di controllo

1. Lo schema
2. Nell'introduzione viene presentato l'argomento o l'idea centrale, nella conclusione viene presentato un riepilogo.
3. In ogni sottoargomento viene descritta l'idea centrale di un paragrafo.
4. Le informazioni di supporto e i dati dettagliati relativi a un sottoargomento sono riportati al di sotto del sottoargomento e ogni informazione è elencata separatamente.
5. Quando per un sottoargomento viene inserito un elenco di informazioni di supporto, le informazioni elencate devono essere almeno due. In caso contrario, l'informazione viene inclusa nel sottoargomento.
6. Il documento
7. Il documento segue l'organizzazione dello schema.
8. Ogni paragrafo del documento corrisponde a un sottoargomento dello schema e include le informazioni e i dati dettagliati elencati per i sottoargomenti.
9. Ogni paragrafo include una frase principale che riassume l'idea centrale del paragrafo stesso.
10. Ogni frase inizia con una lettera maiuscola.
11. Ogni frase termina con un punto, un punto interrogativo o un punto esclamativo.
12. L'ortografia delle parole è corretta.
13. Non vi sono vocaboli mancanti.
14. Opere citate
15. Per ogni fonte è presente un riferimento specifico nel documento. Includere solo le fonti menzionate nel documento.
16. Per ogni voce è stato utilizzato il formato corretto in base al tipo di riferimento.
17. Le voci sono in ordine alfabetico, in base al cognome dell'autore.

Suggerimenti per la stesura della relazione

1. Creare una pianificazione
2. Identificare le attività necessarie.
3. Disporre le attività in base all'ordine in cui dovranno essere eseguite.
4. Valutare il tempo necessario per ogni attività. Calcolare un tempo sufficiente per la stesura e le modifiche.
5. Identificare la data di consegna della relazione e creare una pianificazione in cui siano indicate le attività da eseguire ogni giorno per rispettare la scadenza.
6. Rendere la relazione più interessante
7. Utilizzare grafici e diagrammi per illustrare i concetti.
8. Aggiungere un'immagine, una foto o un disegno.
9. Includere una mappa.
10. Cercare una citazione e utilizzarla per sostenere le idee espresse.
11. Dare la giusta importanza a ogni parola
12. Scegliere parole che possano essere comprese dai lettori. Ricordare che l'obiettivo è quello di comunicare le proprie idee ai lettori.
13. Evitare gli stereotipi.
14. Utilizzare un dizionario dei sinonimi per sostituire le parole utilizzate troppo di frequente e cercare nuovi modi per esprimere le idee.