Hilo

Everyone wanna be a Hilo

Object Design Document

Riferimento	
Versione	0.4
Data	19/12/2020
Destinatario	Prof.ssa F. Ferrucci
Presentato da	**H**ermann Senatore, **I**van Carmine Adamo, **L**orenzo Criscuolo, **O**razio Cesarano
Approvato da	

Revision History

DATA	Versione	Cambiamenti	Autori
19/12/2020	0.1	Prima Stesura	[tutti]
21/12/2020	0.2	Aggiunto par 1.4, 1.5	[Lorenzo, Orazio]
22/12/2020	0.3	Aggiunte interfacce	[tutti]
15/01/2020	0.4	Aggiunto class diagram finale e glossario	[tutti]

Sommario

1. Introduzione

1.1. Object Design Trade Off

1.1.1. Usabilità vs Funzionalità

Il Sistema dovrà prediligere l'usabilità a discapito delle funzionalità previste nella fase di Analisi in quanto risulta più prioritario fornire un sistema user friendly a discapito di operazioni superficiali.

1.1.2. Costo vs Robustezza

Il Sistema sarà sviluppato in modo robusto a discapito dei costi in quanto essendo prettamente di uso medico può essere considerato "Mission Critical". Per questo motivo si preferisce sostenere costi maggiori al fine di ottenere un sistema robusto.

1.1.3. Efficienza vs Portabilità

Il Sistema dovrà favorire una maggiore efficienza a discapito della portabilità. Questa scelta nasce dall'esigenza di avere un sistema snello e in grado di eseguire operazioni nel miglior modo e nel minor tempo possibile al fine di suscitare fiducia negli utenti finali del sistema.

1.1.4. Sviluppo Rapido vs Funzionalità

Il Sistema sarà sviluppato con un minor di funzionalità per favorire uno sviluppo rapido, in quanto sono presenti delle deadline e non è presente abbastanza tempo per implementare anche tutte le funzionalità ritenute meno importanti.

1.1.5. Costo vs Riusabilità

Il Sistema proposto, essendo realizzato ex novo, non avrebbe senso parlare di riutilizzo di componenti già esistenti. Per tanto non si può ignorare la necessità di sostenere costi maggiori per lo sviluppo

1.1.6. Tempo di Risposta vs Affidabilità

Il Sistema dovrà garantire una maggiore affidabilità a discapito del tempo di risposta su operazioni critiche in quanto deve essere ridotta al minimo la possibilità di introdurre errori o incongruenze nei dati dovute alla gestione della concorrenza, ecc.

1.2. Components Off the Shelf (COTS)

Il Sistema utilizzerà I seguenti componenti off the shelf:

- Bootstrap, un framework per aiutare lo sviluppo delle interfacce grafiche che utilizza HTML, CSS e JS;
- Thymeleaf, un framework per Spring da utilizzare per il frontend che semplifica la gestione dinamica delle pagine;
- Spring, un framework scritto in Java che semplifica la gestione lato back end e in generale aiuta nello sviluppo di applicazioni basate su MVC;
- JUnit, un framework per agevolare lo unit testing per Java;
- Katalon, un plugin per browser per agevolare system testing;
- Apache Tomcat, un web server con annesso application container per applicazioni scritte in Java;
- DeepLearning4J, una libreria Java che permette la costruzione di un modello di rete neurale a partire da un modello di IA preaddestrato;
- Datavec, per il caricamento delle immagini da dare in pasto al modello di rete neurale;

- ND4J, per effettuare preprocessing sulle immagini prese in input e gestire i dati di output forniti dal modello;

1.3. Interfaces, Documentation, Guidelines

1.3.1. Classi e Interfacce Java

Lo stile di scrittura di codice Java rispetterà gli standard definiti da Google (consultabili al link https://google.github.io/styleguide/javaguide.html). Come altre style guides per la programmazione, le problematiche trattate non riguardano esclusivamente questioni estetiche di formattazione, ma anche altri tipi di convenzioni o standard di codifica. Il documento, comunque, si focalizza principalmente sulle regole definite *hard-and-fast* tra cui:

1. Con il termine classe s'intende una classe "ordinaria", una enum class, un'interfaccia od un'annotazione (es. @interface);
2. Con il termine membro (di una classe) s'intende una classe innestata, un campo, un metodo o costruttore, ossia tutti i contenuti top-level di una classe (eccezion fatta per inizializzatori e commenti);
3. Il termine commento si riferisce sempre a commenti implementativi. Non si utilizzerà la frase "commenti di documentazione", bensì il termine comune "Javadoc".

Di seguito vengono poste alcune regole di formattazione:

1. Le parentesi graffe sono utilizzate con gli statement if, else, for, do e while, anche quando il corpo è vuoto o contiene un singolo statement.
2. Le parentesi sfruttano lo stile definito da Kernighan e Ritchie ("Egyptian brackets") per blocchi non-vuoti e costrutti block-like:
   - nessuna interruzione di riga (line break) prima di aprire la parentesi;
   - un line break dopo l'apertura della parentesi;
   - un line break prima di chiudere la parentesi;
   - un line break dopo la chiusura della parentesi solo se la parentesi chiude uno statement o termina il corpo di un metodo, costruttore o classe.
3. I package sono tutti in lowercase, con parole consecutive semplicemente concatenate senza underscore
4. I nomi delle classi sono scritti in UpperCamelCase. Essi sono tipicamente sostantivi o locuzioni che fungono da sostantivo. I metodi sono scritti in lowerCamelCase, i cui nomi sono tipicamente verbi o locuzioni verbali, come anche i campi (che non sono final) sono scritti in lowerCamelCase con nomi che sono tipicamente sostantivi.
5. Le classi di test hanno nomi che iniziano con il nome della classe testata e terminano con "Test". Ad esempio, "HashTest" o "HashIntegrationTest".

Per ulteriori regole, consultare il link posto all'inizio di questo paragrafo.

1.3.2. Pagine HTML

Le pagine HTML dovranno rispettare gli standard proposti da Google. L'URL per trovare la guida è fornito nel paragrafo 1.3.2 e per le pagine sarà utilizzato HTML5.

1.3.3. Script JavaScript

Gli script scritti in JavaScript devono rispettare gli standard di Google (URL si trova sempre nel paragrafo 1.3.2).

1.3.4. Fogli di Stile CSS

I fogli di stile CSS dovranno anch'essi rispettare gli standard di Google (URL nel paragrafo 1.3.2) per essere conformi allo stile delle pagine HTML e verrà utilizzato. Sarà utilizzato, ove possibile, un validatore di CSS per eliminare proprietà inutili che non hanno effetto sulla pagina e quindi rendere il codice CSS più leggibile e snello. Come validatore, si utilizzerà il W3C Validator, che si trova al link https://jigsaw.w3.org/css-validator/

1.3.5. DB SQL

Il DB deve rispettare le seguenti convenzioni:

1. Il nome delle tabelle deve essere composto solo da lettere minuscole e, in caso di più parole, deve esserci un underscore tra di esse piuttosto che di uno spazio;
2. I nomi degli attributi devono essere scritti in lowerCamelCase e devono essere sostantivi tratti dal dominio del problema o, in caso di chiavi esterne, sostantivi che rendano immediata la comprensione delle relazioni;

1.4. Design Pattern e Architectural Pattern

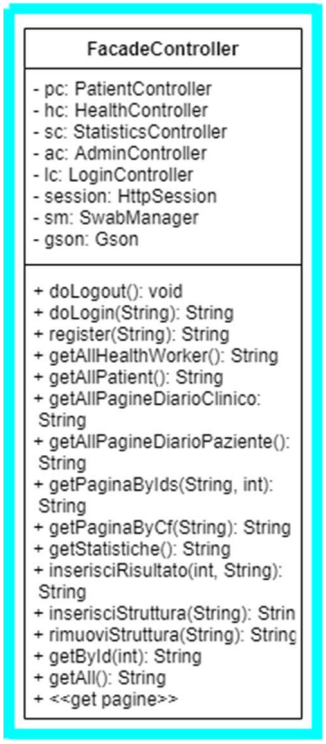
1.4.1. MVC (Model View Control)

Il Model-View-Control (MVC) è un pattern utilizzato in programmazione per dividere il codice in blocchi dalle funzionalità ben distinte. I blocchi principali sono tre:

- **Model:** è il blocco contente la logica di business e l'interazione con i dati persistenti, esponendo alla View ed al Control rispettivamente le funzionalità per l'accesso e aggiornamento dei dati. Il model può inoltre avere il compito di notificare alle View eventuali cambiamenti richiesti dal Control in modo tale da mostrare sempre dati aggiornati;
- **View:** è il blocco che ha il compito di occuparsi della presentazione dei dati e delle possibili interazioni con essi; questo implica l'implementazione dell'interfaccia grafica con la quale l'utenza interagisce con il sistema. Ogni UI può essere formata da più pagine ognuna mostrante dati e operazioni diverse;
- **Control:** è il blocco che si occupa di mappare le azioni svolte dal blocco View sul blocco Model. Questo significa che questo blocco fa da intermediario tra i due blocchi ed è responsabile di interpretare correttamente le operazioni richieste al fine di richiamare le corrette funzioni di business

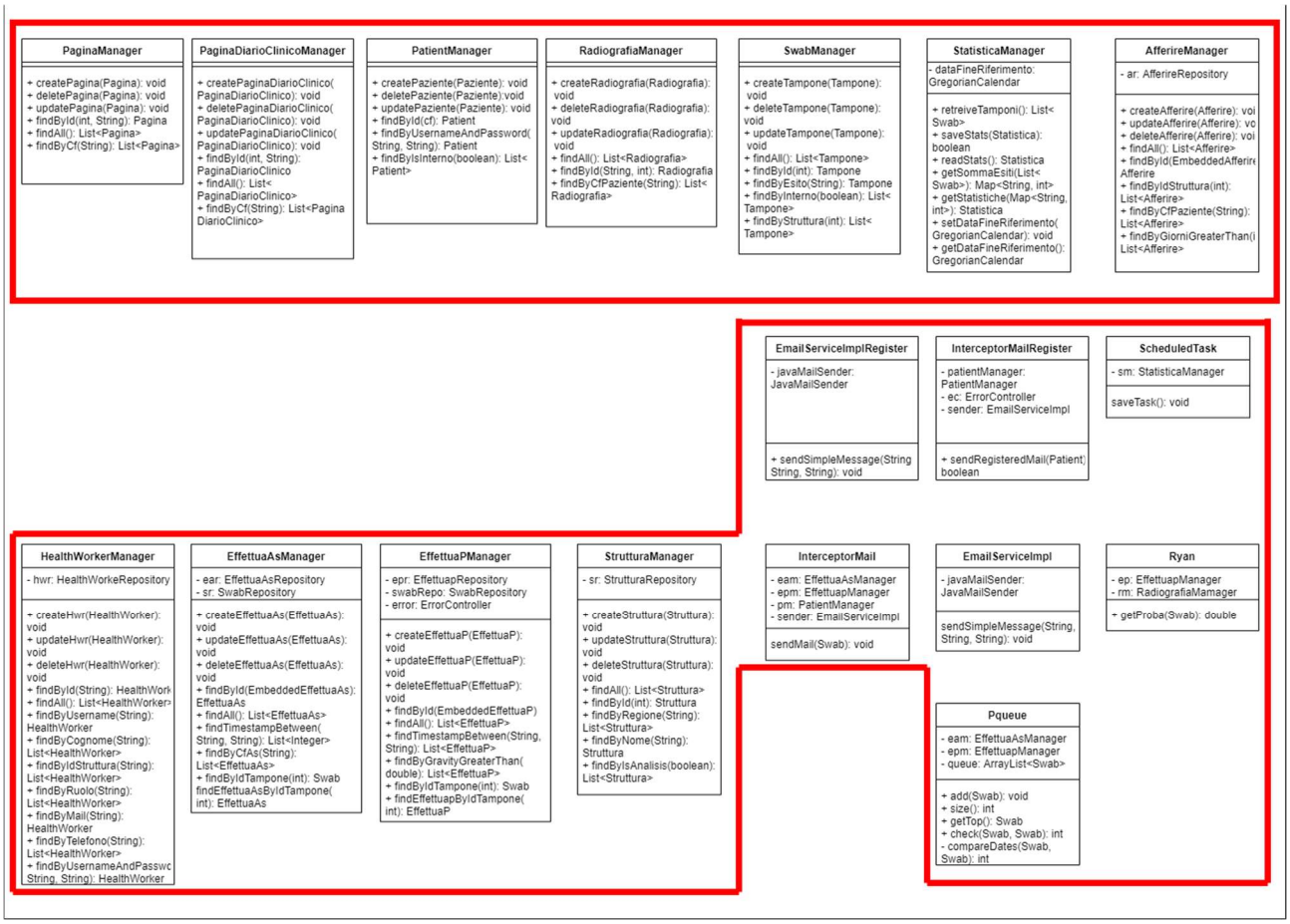
1.4.2. Façade

Il design pattern façade fornisce l'interfaccia per accedere ad un gruppo di oggetti che compone un sottosistema. Questo tipo di pattern dovrebbe essere utilizzato da ogni sottosistema di un software: esso permette di identificare i servizi offerti dal sottosistema e fornisce un'architettura chiusa. Fornendo l'interfaccia rende il sottosistema più facile da utilizzare e allo stesso tempo lo rende più sicuro e disaccopiato.



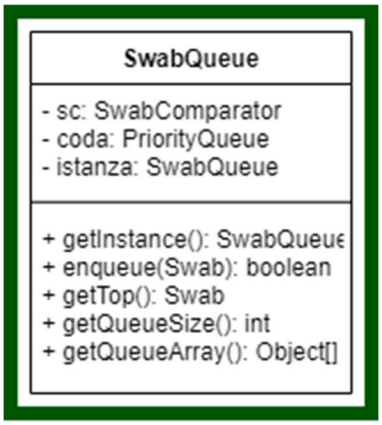
1.4.3. Inversion of Control

Il design pattern IoC, in particolare la “Context and Dependency Injection”, permette di non curarsi delle dipendenze in quanto per mezzo di un container è possibile soddisfarle in automatico, fornendo supporto attivo alla stesura del codice. La risoluzione delle dipendenze avviene tramite quella che viene chiamata DI (Dependency Injection), ovvero annotare dei punti detti “di iniezione” dove abbiamo necessità di risolvere una dipendenza. Questo pattern è peculiare del framework Spring, sul quale il sistema si basa interamente.



1.4.4. Singleton

Il compito del design pattern Singleton è quello di assicurare che una data classe sia istanziata una sola volta e di fornire un accesso a quest'ultima a tutte le classi che lo richiedono.



1.5. Definizioni, Acronimi e Abbreviazioni

- User friendly: di semplice utilizzo e comprensione per l'utente finale del sistema
- Mission Critical: fondamentale nel contesto del sistema; che è necessario per la riuscita del progetto;
- ex novo: realizzato da zero;
- application container: è un software che si occupa della gestione completa del ciclo di vita di tutte le classi all'interno e della gestione delle interazioni con l'esterno del sistema;

- framework: un insieme di API che svolgono compiti onerosi per lo sviluppatore e che servono per velocizzare e facilitare lo sviluppo del software;
- style guides: linee guida per essere conforme ad uno stile che può essere di scrittura, di progettazione, ecc.;
- UI: "User Interface", letteralmente interfaccia utente;
- event-driven: tipologia di sistema "basato su eventi" dove le operazioni vengono azionate al verificarsi di determinati eventi;
- HTML: acronimo di HyperText Markup Language, un linguaggio usato per definire la struttura di una pagina web;
- CSS: acronimo di "Cascading Style Sheets", un linguaggio di usato per definire lo stile e la formattazione di una pagina web;
- JS: acronimo di JavaScript, un linguaggio di scripting utilizzato nelle pagine web per fornire dinamicità e logica a queste ultime;
- lowerCamelCase: tecnica di naming che prevede la scrittura di più parole senza spazi e delimitando l'inizio di una nuova parola con una maiuscola. La prima lettera della prima parola è in minuscolo;
- UpperCamelCase: tecnica di naming uguale alla precedente con l'unica differenza riguardo la prima lettera della prima parola che in questo caso è maiuscola;
- Statement: singola istruzione generica di un linguaggio di programmazione;

2. Packages

2.1. Model

Le classi che faranno parte di questo package saranno:

- Package swabmanagement:
  - SwabQueue;
  - SwabManager;
  - Swab;
  - Ryan (modulo IA);
  - Pqueue;
  - EmbeddedEffettuaP;
  - EmbeddedEffettuaAs;
  - EffettuapManager;
  - EffettuaP;
  - InterceptorMail;
  - EmailServiceImpl;
- Package statisticsmanagement:
  - StatisticaManager;
  - Statistica;
  - ScheduledTask;
- Package patientmanagement:
  - RadiografiaManager;
  - Radiografia;
  - PatientManager;
  - Patient;
  - PaginaManager;
  - PaginaDiarioClinicoManager;
  - PaginaDiarioClinico;

    - Pagina;
    - InterceptorMailRegister;
    - EmailServiceImplRegister;
- Package healthworkermanagement:
    - HealthWorker;
    - HealthWorkerManager;
- Package adminmanagement:
    - StrutturaManager;
    - Struttura;
    - AfferireManager;
    - Afferire;

2.2. View

- Situata nella cartella resources/templates:
    - aggiungi-operatore;
    - aggiungi-paziente;
    - aggiungi-radiografia;
    - aggiungi-tampone;
    - HomePage;
    - Login;
    - navbar;
    - operatore-homepage;
    - recover;
    - ricerca-tampone-utente;
    - Statistiche;
    - Su-di-noi;

2.3. Controller

- AdminController;
- PatientController;
- HealthController;
- ErrorController;
- StatisticsController;
- SwabController;
- FacadeController;
- LoginController;

3. Class Interfaces

Nome classe	SwabQueue
Descrizione	Questa classe permette di gestire la coda dei tamponi da analizzare
Pre-Condizione	context SwabQueue::enqueue(s) pre: not contains(s)
Post-Condizione	context SwabQueue::enqueue(s) post: contains(s) context SwabQueue::getTop(s) post: not contains(s)
Invarianti	

Nome classe	SwabManager
Descrizione	Questa classe permette di gestire i tamponi
Pre-Condizione	Context SwabManager::createSwab(s) pre: findById(s.getId()) = null Context SwabManagere::updateSwab(s) pre: not findById(s.getId()) = null
Post-Condizione	Context SwabManager::createSwab(s) post: findById(s.getId) = swab Context SwabManager::updateSwab(s) post: findById(s.getId()) = swab Context SwabManager::deleteSwab(s) post: findById(s.getId()) = null
Invarianti	

Nome classe	EffettuapManager
Descrizione	Questa classe permette di gestire i tamponi che sono stati effettuati da pazienti
Pre-Condizione	Context EffettuapManager::createEffettuaP(ep) pre: findById(ep.id()) = null Context effettuapManager::updateEffettuaP(ep) pre: not findById(ep.id()) = null Context effettuapManager::findByTimestampBetween(t1, t2) pre: t1 < t2
Post-Condizione	Context EffettuapManager::createEffettuaP(ep) post: findById(ep.id()) = ep Context effettuapManager::updateEffettuaP(ep) Post: findById(ep.id()) = ep
Invarianti	

Nome classe	EffettuaAsManager
Descrizione	Questa classe permette di gestire i tamponi che sono stati effettuati da operatori sanitari
Pre-Condizione	Context EffettuaAsManager::createEffettuaAs(as) pre: findById(as.id()) = null Context effettuaAsManager::updateEffettuaAs(as) pre: not findById(as.id()) = null Context effettuaAsManager::findByTimestampBetween(t1, t2) pre: t1 < t2
Post-Condizione	Context EffettuaAsManager::createEffettuaAs(as) post: findById(as.id()) = as Context effettuaAsManager::updateEffettuaAs(as) Post: findById(as.id()) = as
Invarianti	

Nome classe	StatisticaManager
Descrizione	Questa classe permette di gestire le statistiche periodiche dei contagi
Pre-Condizione	
Post-Condizione	
Invarianti	

Nome classe	RadiografiaManager
Descrizione	Questa classe permette di gestire le radiografie
Pre-Condizione	Context RadiografiaManager::createRadiografia(r) pre: findById(r.getId()) = null Context RadiografiaManager::updateRadiografia(r) pre: not findById(r.getId()) = null
Post-Condizione	Context RadiografiaManager::createRadiografia(r) post: findById(r.getId()) = r Context RadiografiaManager::updateRadiografia(r) post: findById(r.getId()) = r Context RadiografiaManager::deleteRadiografia(r) post: findById(r.getId()) = null
Invarianti	

Nome classe	PatientManager
Descrizione	Questa classe permette di gestire i pazienti afferenti ad una struttura
Pre-Condizione	Context PatientManager::createPatient(p) pre: findById(p.getCf()) = null Context PatientManager::updatePatient(p) pre: not findById(p.getCf()) = null
Post-Condizione	Context PatientManager::createPatient(p) post: findById(p.getCf()) = p Context PatientManager::updatePatient(p) post: findById(p.getCf()) = p Context PatientManager(p)::deletePatient(p) post: findById(p.getCf()) = null
Invarianti	

Nome classe	PaginaManager
Descrizione	Questa classe permette di gestire delle pagine di un diario clinico compilato da un paziente
Pre-Condizione	Context PaginaManager::createPagina(p) pre: findById(p.getId()) = null Context PaginaManager::updatePagina(p) pre: not findById(p.getId()) = null

Post-Condizione	Context PaginaManager::createPagina(p) post: findById(p.getId()) = p Context PaginaManager::updatePagina(p) post: findById(p.getId()) = p Context PaginaManager::deletePagina(p) post: findById(p.getId()) = null
Invarianti	

Nome classe	**PaginaDiarioClinicoManager**
Descrizione	Questa classe permette di gestire delle pagine di una cartella clinica associata ad un paziente
Pre-Condizione	Context PaginaDiarioClinicoManager::createPaginaDiarioClinico(p) pre: findById(p.getId()) = null Context PaginaDiarioClinicoManager::updatePaginaDiarioClinico(p) pre: not findById(p.getId()) = null
Post-Condizione	Context PaginaDiarioClinicoManager::createPaginaDiarioClinico(p) post: findById(p.getId()) = p Context PaginaDiarioClinicoManager::updatePaginaDiarioClinico(p) post: findById(p.getId()) = p Context PaginaDiarioClinicoManager::deletePaginaDiarioClinico(p) post: findById(p.getId()) = null
Invarianti	

Nome classe	**HealthWorkerManager**
Descrizione	Questa classe permette di gestire gli operatori sanitari afferenti alla struttura
Pre-Condizione	Context HealthWorkerManager::createHwr(h) pre: findById(h.getCf()) = null Context HealthWorkerManager::updateHwr(h) pre: not findById(h.getCf()) = null
Post-Condizione	Context HealthWorkerManager::createHwr(h) post: findById(h.getCf()) = h Context HealthWorkerManager::updateHwr(h) post: findById(h.getCf()) = h Context HealthWorkerManager::deleteHwr(h) post: findById(h.getCf()) = null
Invarianti	

Nome classe	**StrutturaManager**

Descrizione	Questa classe permette di gestire le strutture che utilizzano la piattaforma
Pre-Condizione	Context StrutturaManager::createStruttura(s) pre: findById(s.getId()) = null Context StrutturaManager::updateStruttura(s) pre: not findById(s.getId()) = null
Post-Condizione	Context StrutturaManager::createStruttura(s) post: findById(s.getId()) = s Context StrutturaManager::updateStruttura(s) post: findById(s.getId()) = s Context StrutturaManager::deleteStruttura(s) post: findById(s.getId()) = null
Invarianti	

Nome classe	**AfferireManager**
Descrizione	Questa classe permette di gestire le relazioni di afferenza tra pazienti e strutture
Pre-Condizione	Context AfferireManager::createAfferire(a) pre: findById(a.getId()) = null Context AfferireManager::updateAfferire(a) pre: not findById(a.getId()) = null
Post-Condizione	Context AfferireManager::createAfferire(a) post: findById(a.getId()) = a Context AfferireManager::updateAfferire(a) post: findById(a.getId()) = a Context AfferireManager::deleteAfferire(a) post: findById(a.getId()) = a
Invarianti	

Nome classe	**SwabComparator**
Descrizione	Questa classe specifica il criterio di ordinamento della coda dei tamponi
Pre-Condizione	Context SwabComparator::compare(s1, s2) pre: not s1 = null && not s2 = null
Post-Condizione	
Invarianti	

Nome classe	**Swab**
Descrizione	Questa classe rappresenta un tampone all'interno del sistema
Pre-Condizione	
Post-Condizione	
Invarianti	Context Swab inv: not id = null

Nome classe	**Ryan**

Descrizione	Questa classe implementa il modulo di IA previsto per la piattaforma
Pre-Condizione	
Post-Condizione	
Invarianti	

Nome classe	**EmbeddedEffettuaP**
Descrizione	Questa classse modella la chiave primaria dell'entità effettuaP
Pre-Condizione	
Post-Condizione	
Invarianti	

Nome classe	**EmbeddedEffettuaAs**
Descrizione	Questa classe modella la chiave primaria dell'entità effettuaAs
Pre-Condizione	
Post-Condizione	
Invarianti	

Nome classe	**EffettuaP**
Descrizione	Questa classe modella la relazione EffettuaP, ovvero un tampone effettuato da un paziente
Pre-Condizione	Context EffettuaP::setCfP(cfP) pre: cfP.matches("^[A-Z]{6}\d{2}[A-Z]\d{2}[A-Z]\d{3}[A-Z]$")
Post-Condizione	
Invarianti	

Nome classe	**EffettuaAs**
Descrizione	Questa classe modella la relazione EffettuaAs, ovvero un tampone effettuato da un agente sanitario
Pre-Condizione	Context EffettuaAs::setCfAs(cfAs) pre: cfAs.matches("^[A-Z]{6}\d{2}[A-Z]\d{2}[A-Z]\d{3}[A-Z]$")
Post-Condizione	
Invarianti	

Nome classe	**EffettuaAsRepository**
Descrizione	Questa classe è necessaria al framework Spring per poter definire le query per EffettuaAs
Pre-Condizione	
Post-Condizione	

Invarianti	

Nome classe	**EffettuapRepository**
Descrizione	Questa classe è necessaria al framework Spring per poter definire le query per Effettuap
Pre-Condizione	
Post-Condizione	
Invarianti	

Nome classe	**SwabRepository**
Descrizione	Questa classe è necessaria al framework Spring per poter definire le query per Swab
Pre-Condizione	
Post-Condizione	
Invarianti	

Nome classe	**InterceptorMail**
Descrizione	Questa classe permette di compilare una e-mail ad un destinatario dell'esito di un tampone
Pre-Condizione	Context InterceptorMail::sendMail(s) pre: not findById(s) = null and not getCf() = null and not getMail() = null
Post-Condizione	
Invarianti	

Nome classe	**EmailServiceImpl**
Descrizione	Questa classe permette di inviare effettivamente le e-mail compilate da InterceptorMail
Pre-Condizione	Context EmailServiceImpl::sendSimpleMessage(to, subject, text) pre: not to = null and not subject = null and not text = null
Post-Condizione	
Invarianti	

Nome classe	**Statistica**
Descrizione	Questa classe modella una statistica dei contagi da pubblicare
Pre-Condizione	
Post-Condizione	
Invarianti	

Nome classe	**ScheduledTask**

Descrizione	Questa classe implementa un task periodico che si occupa di salvare le statistiche su un file
Pre-Condizione	
Post-Condizione	
Invarianti	

Nome classe	**Radiografia**
Descrizione	Questa classe modella una radiografia effettuata da un paziente interno
Pre-Condizione	Context Radiografia pre: not cfPaziente = null
Post-Condizione	
Invarianti	

Nome classe	**Patient**
Descrizione	Questa classe modella un paziente afferente ad una struttura
Pre-Condizione	Context Patient pre: cf.matches("^[A-Z]{6}\d{2}[A-Z]\d{2}[A-Z]\d{3}[A-Z]$")
Post-Condizione	
Invarianti	

Nome classe	**PaginaDiarioClinico**
Descrizione	Questa classe modella una pagina di cartella clinica associata ad un paziente
Pre-Condizione	Context PaginaDiarioClinico pre: cfPaziente.matches("^[A-Z]{6}\d{2}[A-Z]\d{2}[A-Z]\d{3}[A-Z]$")
Post-Condizione	
Invarianti	

Nome classe	**Pagina**
Descrizione	Questa classe modella una pagina di diario clinico compilata da un paziente afferente
Pre-Condizione	Context Pagina pre: cfPaziente.matches("^[A-Z]{6}\d{2}[A-Z]\d{2}[A-Z]\d{3}[A-Z]$")
Post-Condizione	
Invarianti	

Nome classe	**EmbeddedRadiografia**
Descrizione	Questa classe, necessaria per il framework Spring, modella la chiave primaria di Radiografia
Pre-Condizione	
Post-Condizione	

Invarianti	Context EmbeddedRadiografia inv: not cfpaziente = null and not numero = null

Nome classe	EmbeddedPaginaDiarioClinico
Descrizione	Questa classe, necessaria per il framework Spring, modella la chiave primaria di PaginaDiarioClinico
Pre-Condizione	
Post-Condizione	
Invarianti	Context EmbeddedPaginaDiarioClinico inv: not numero = null and not cfpaziente = null

Nome classe	EmbeddedPagina
Descrizione	Questa classe, necessaria per il framework Spring, modella la chiave primaria di Pagina
Pre-Condizione	
Post-Condizione	
Invarianti	Context EmbeddedPagina inv: not numero = null and not cfpaziente = null

Nome classe	PaginaDiarioClinicoRepository
Descrizione	Questa classe è necessaria al framework Spring per poter definre le query su PaginaDiarioClinico
Pre-Condizione	
Post-Condizione	
Invarianti	

Nome classe	PaginaRepository
Descrizione	Questa classe è necessaria al framework Spring per poter definre le query su Pagina
Pre-Condizione	
Post-Condizione	
Invarianti	

Nome classe	PatientRepository
Descrizione	Questa classe è necessaria al framework Spring per poter definire le query su Patient
Pre-Condizione	
Post-Condizione	
Invarianti	

Nome classe	RadiografiaRepository
Descrizione	Questa classe è necessaria al framework Spring per poter definire le query su Radiografia

Pre-Condizione	
Post-Condizione	
Invarianti	

Nome classe	**InterceptorMailRegister**
Descrizione	Questa classe permette di compilare una mail con username e password a nuovi pazienti che effettuano un tampone
Pre-Condizione	Context InterceptorMailRegister::sendRegisteredMail(p) pre: not p.getCf() = null and not p.getMail() = null
Post-Condizione	
Invarianti	

Nome classe	**EmailServiceImplRegister**
Descrizione	Questa classe permette di inviare una mail compilata da InterceptorMailRegister
Pre-Condizione	Context EmailServiceImplRegister::sendSimpleMessage(to, subject, text) pre: not to = null and not subject = null and not text = null
Post-Condizione	
Invarianti	

Nome classe	**HealthWorker**
Descrizione	Questa classe modella un operatore sanitario afferente ad una struttura
Pre-Condizione	
Post-Condizione	
Invarianti	Context HealthWorker inv: not cf = null

Nome classe	**HealthWorkerRepository**
Descrizione	Questa classe è necessaria per il framework Spring per definire le query per HealthWorker
Pre-Condizione	
Post-Condizione	
Invarianti	

Nome classe	**Afferire**
Descrizione	Questa classe modella la relazione di afferenza tra paziente e una struttura
Pre-Condizione	
Post-Condizione	
Invarianti	Context inv: not cfPaziente = null

Nome classe	Struttura
Descrizione	Questa classe modella una struttura che utilizza la piattaforma
Pre-Condizione	
Post-Condizione	
Invarianti	

Nome classe	AfferireRepository
Descrizione	Questa classe è necessaria per il framework Spring per definire le query per Afferire
Pre-Condizione	
Post-Condizione	
Invarianti	

Nome classe	StrutturaRepository
Descrizione	Questa classe è necessaria per il framework Spring per definire le query per Struttura
Pre-Condizione	
Post-Condizione	
Invarianti	

Nome classe	EmbeddedAfferire
Descrizione	Questa classe è necessaria per il framework Spring per definire la chiave primaria di Afferire
Pre-Condizione	
Post-Condizione	
Invarianti	Context EmbeddedAfferire inv: not cfPaziente = null

Nome classe	StatisticsController
Descrizione	Questa classe si occupa di gestire le operazioni riguardanti la pubblicazione delle statistiche
Pre-Condizione	
Post-Condizione	
Invarianti	

Nome classe	PatientController
Descrizione	Questa classe si occupa di gestire le operazioni riguardanti i pazienti
Pre-Condizione	
Post-Condizione	
Invarianti	

Nome classe	LoginController
Descrizione	Questa classe si occupa di eseguire le operazioni di login e logout
Pre-Condizione	
Post-Condizione	
Invarianti	

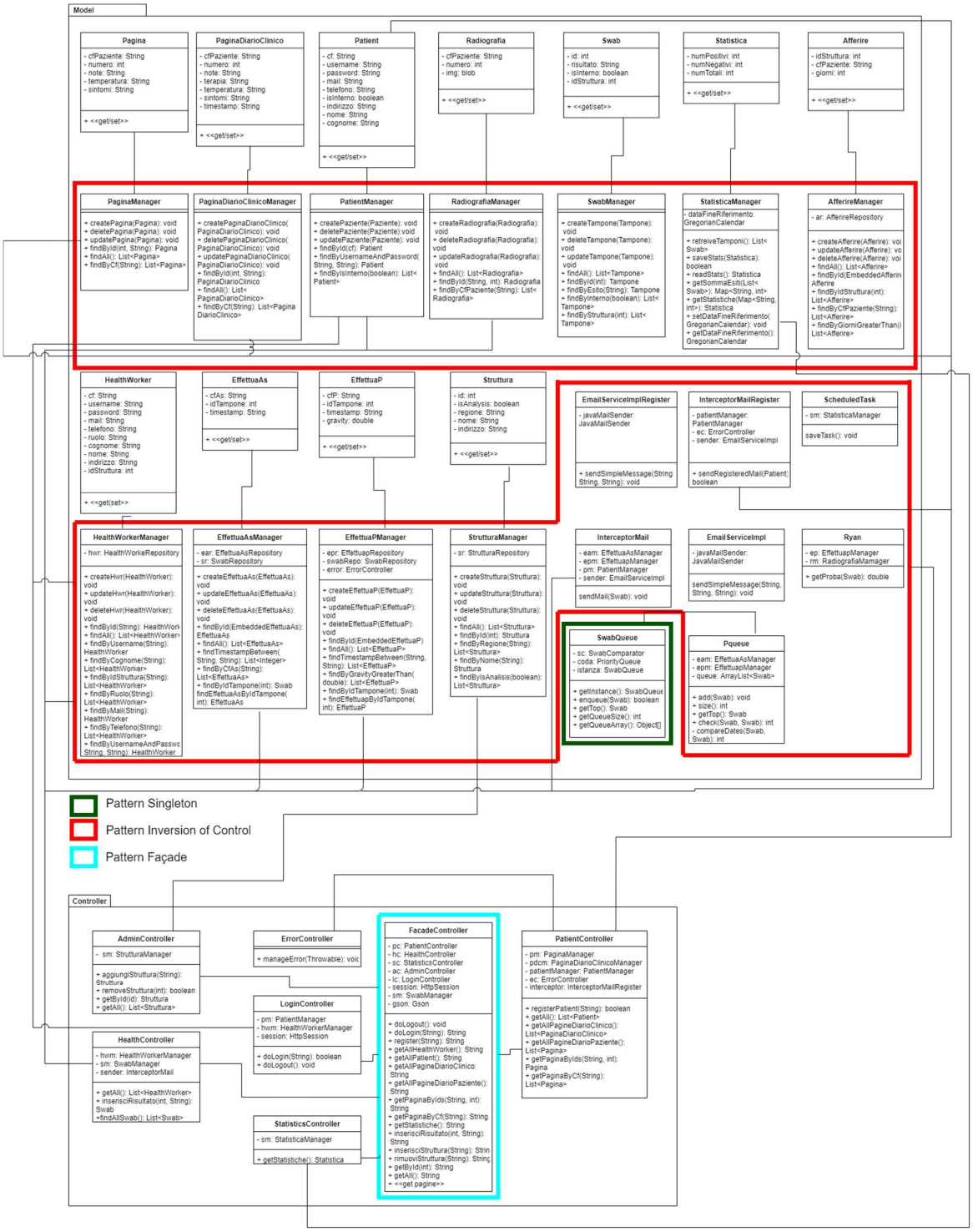
Nome classe	HealthController
Descrizione	Questa classe si occupa di gestire le operazioni effettuabili da un operatore sanitario
Pre-Condizione	
Post-Condizione	
Invarianti	

Nome classe	FacadeController
Descrizione	Questa classe si occupa di intercettare le richieste provenienti dai client e le smista ai controller corretti
Pre-Condizione	
Post-Condizione	
Invarianti	

Nome classe	ErrorController
Descrizione	Questa classe si occupa di gestire gli errori in generale
Pre-Condizione	
Post-Condizione	
Invarianti	

Nome classe	AdminController
Descrizione	Questa classe si occupa di gestire tutte le operazioni possibili dell'admin
Pre-Condizione	
Post-Condizione	
Invarianti	

4. Class Diagram



5. Glossario

- Deadline, ovvero una scadenza;
- Off The Shelf, solitamente preceduto da “Component”, è un qualcosa (un componente) già pronto all’uso;
- DB, acronimo di DataBase ovvero “Base di Dati” che è un archivio persistente;
- SQL, acronimo di “Structured Query Language”, che è un linguaggio dichiarativo utilizzato per interagire con DataBase di tipo Relazionale;
- UI, acronimo di “User Interface”, ovvero un’interfaccia grafica con la quale un utente può interagire;