

**O**bject

**D**esign

**D**ocument

|  |  |
| --- | --- |
| Riferimento |  |
| Versione | 0.1 |
| 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] |

Sommario

[1. Introduzione 2](#_Toc59286893)

[1.1. Object Design Trade Off 2](#_Toc59286894)

[1.2. Components Off the Shelf (COTS) 2](#_Toc59286895)

[1.3. Interfaces, Documentation, Guidelines 3](#_Toc59286896)

[1.3.1. Classi e Interfacce Java 3](#_Toc59286897)

[1.3.2. Pagine Lato Server 3](#_Toc59286898)

[1.3.3. Pagine HTML 3](#_Toc59286899)

[1.3.4. Script JavaScript 3](#_Toc59286900)

[1.3.5. Fogli di Stile CSS 3](#_Toc59286901)

[1.3.6. DB SQL 3](#_Toc59286902)

[1.4. Design Pattern e Architectural Pattern 3](#_Toc59286903)

[1.4.1. MVC (Model View Control) 3](#_Toc59286904)

[1.4.2. Da decidere 1 3](#_Toc59286905)

[1.4.3. Da decider 2 3](#_Toc59286906)

[1.4.4. Da decider 3 3](#_Toc59286907)

[1.5. Definizioni, Acronimi e Abbreviazioni 3](#_Toc59286908)

[2. Packages 3](#_Toc59286909)

[2.1. Interface 3](#_Toc59286910)

[2.2. Model 3](#_Toc59286911)

[2.3. View 3](#_Toc59286912)

[2.4. Control 3](#_Toc59286913)

[3. Class Interfaces 3](#_Toc59286914)

[4. Class Diagram 3](#_Toc59286915)

[5. Glossario 3](#_Toc59286916)

# Introduzione

## Object Design Trade Off

### 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.

### 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.

### 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.

### 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.

### 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

### 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.

## 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;
* Vue.js, un framework per JS per semplificare la validazione dell’input e in generale del front end;
* 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;
* Jython, una libreria Java che permette l’esecuzione di codice Python all’interno di una classe Java;

## Interfaces, Documentation, Guidelines

### 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.

1. I package sono tutti in lowercase, con parole consecutive semplicemente concatenate senza underscore
2. 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.
3. 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.

### Pagine Vue

Le pagine Vue quando vengono visualizzate, essendo composte da HTML e JS, devono rispettare gli standard di codifica proposti da Google sia per HTML che per JS per entrambe le sintassi. I due standard sono proposti rispettivamente ai seguenti link:

1. <https://google.github.io/styleguide/htmlcssguide.html>
2. <https://google.github.io/styleguide/jsguide.html>

### Pagine HTML

Le pagine HTML dovranno rispettare gli standard proposti da Google in modo da avere uno stile conforme anche con le pagine Vue. L’URL per trovare la guida è fornito nel paragrafo 1.3.2 e per le pagine sarà utilizzato HTML5.

### Script JavaScript

Gli script scritti in JavaScript devono rispettare gli standard di Google (URL si trova sempre nel paragrafo 1.3.2) in modo da essere conformi allo stile utilizzato per le pagine Vue.

### 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/

### 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;

## Design Pattern e Architectural Pattern

### 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

### 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.

### Observer

Il compito del design pattern Observer è quello di realizzare una gestione event-driven, e quindi fare in modo che delle classi possano ricevere una notifica ogniqualvolta viene apportata una modifica ad una classe “osservata”.

### Singleton

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

## 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;

# Packages

## Interface

## Model

* Swab;
* SwabManagement;
* HealthWorker;
* HealtWorkerManagement
* Patient;
* Struttura;
* Pagina;
* PaginaManagement
* Radiografia;
* RadiografiaManagement;
* PaginaDiarioClinico;
* PaginaDiarioClinicoManagement;
* CodaTamponi;
* ListaTamponi;
* ListaRisultati;
* CodaNotifiche;
* Notifica;
* Effettua\_AS;
* Effettua\_P;
* Afferire;
* IA.

## View

* GUIPaziente(Esiti, FormStato, DiarioClinico, HomePaziente);
* Login;
* Logout;
* GUIOperatore(AggiuntaTampone, aggiungiRadiografia, aggiuntaEsito, homeOperatore, VisualizzaTamponi);
* GUIAdmin(creaAccountPaziente, aggiungiOperatore, visualizzaDatiAfferenti);
* GUIStatistiche(HomePage);
* GUIError;

## Control

* AdminController;
* PatientController;
* HealthController;
* ValidationController;
* ErrorController;
* StatisticsController;
* SwabController;
* FacadeController;

# Class Interfaces

Tampone

# Class Diagram

# Glossario