

ODD

Object Design Document

MedQueue

Afeltra Angelo

Amato Adriano

Fucile Andrea

Rapa Giovanni

Sommario

1. **Introduzione**
   1. Object design trade-offs
      1. Componenti off-the-shelf
      2. Design pattern
   2. Linee guida per la documentazione dell’interfaccia
   3. Definizioni, acronimi e abbreviazioni
   4. Riferimenti
2. **Packages**
   1. Divisione in pacchetti
   2. Organizzazione del codice in file
3. **Interfacce delle classi**
4. **Class diagram**
5. **Glossario**
6. **INTRODUZIONE**
   1. **Object design trade-offs**

Buy vs Build: la necessità di sviluppare un’applicazione web apre alla possibilità di utilizzare una vastissima collezione di framework e librerie utili per la realizzazione del prodotto.

Data la volonta di voler consegnare il prodotto rispettando la data di consegna comunicata inzialmente, abbiamo valutato l’adozione di componenti off-the-shelf per il core dell’applicazione.

E in oltre rilevante sottolineare che le scelte sono state dettate anche dalla facilità di adattamento del team di sviluppo alle nuove componenti: si è infatti cercato il compromesso tra familiarizzare con una tecnologia sconosciuta e la reale utilita della suddetta.

Per quanto riguarda l’applicazione desktop che verrà fornita alla struttura ospedaliera, visto il rapido sviluppo non si utilizzerà nessuna componente off-the-shelf.

* + 1. **Componenti off-the-shelf**

Per motivazioni legate al budget ridotto, si ricorrerà all’utilizzo di un framework per la realizzazione dell’interfaccia utente: Ionic.

Scritto in HTML, CSS e Vue.js, questo framework ha l’obiettivo…. (CONTINUARE).

Questa soluzione si presta alla realizzazione di un’interfaccia completa e minimale, ideale per il servizio che forniremo.

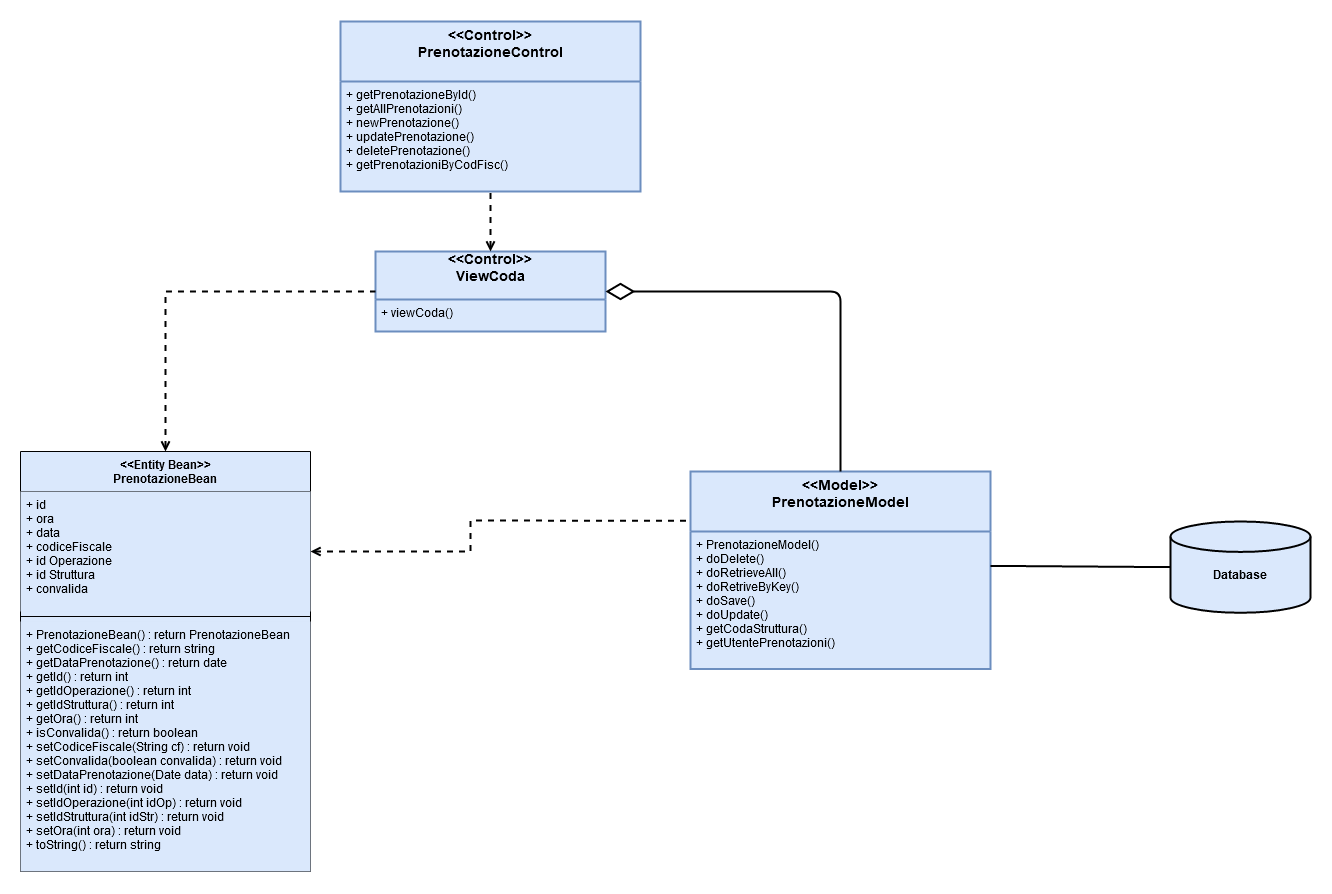
Il back-end farà invece forte affidamento sul framework Spring, nota soluzione nell’ambito delle applicazioni distribuite Java. Composto da un core ben ottimizzato e prodotto con l’obiettivo di ridurre quello che il codice “boilerplate”, consente agli sviluppatori di concentrarsi maggiormente sulla logica di business dell’applicazione, piuttosto che sulla comunicazione tra le varie componenti. Il modulo d’interesse per questo progetto è Spring MVC (che fornisce tutto il meccanismo di controllo del flusso) .

La comunicazione tra front-end e back-end avviene ……(COMPLETARE)

* + 1. **Design patterns**

Per velocizzare lo sviluppo della webapp, abbiamo utilizzato alcuni design patterns che risolvono problemi comuni.

I pattern adottati si integrano correttamente e, pertanto, presentiamo il diagramma che mostra come si relazionano le varie componenti introdotte, mentre rimandiamo ai link presenti nella sezione 1.4 per schemi dettagliati che riguardano i singoli pattern.



* 1. **Linee guida per la documentazione dell’interfaccia**

E richiesto agli sviluppatori di seguire le seguenti linee guida al fine di essere consistenti nell’intero progetto e facilitare la comprensione delle funzionalità di ogni componente.

Nomenclatura delle componenti:

* Nomi delle classi
* Ogni classe deve avere nome in CamelCase
* Ogni classe deve avere nome singolare
* Ogni classe che modella un’entità deve avere per nome un sostantivo che possa associarlo alla corrispondente entità di dominio
* Ogni classe che realizza un servizio offerto via web deve avere nome composto da nome del servizio seguito dal suffisso “Control”
* Nomi dei metodi
* Ogni metodo deve avere nome in lowerCamelCase
* Ogni metodo in cui si può verificare un errore deve segnalare l’errore sollevando un’eccezione
* Nomi delle eccezioni
* Ogni eccezione deve avere nome esplicativo del problema segnalato
* Nomi degli altri sorgenti
* Ogni documento JSP deve avere nome che possa ricondurre al contenuto da essa mostrato

Organizzazione delle componenti:

* Tutte le classi che realizzano un sottoinsieme devono essere racchiuse nello stesso pacchetto Java
* Tutte le componenti che realizzando l’interfaccia grafica devono essere collocate nella directory “Inserire directory”
* Tutte le risorse statiche (fogli di stile, script e immagini) devono essere collocate nella directory “resources”

Organizzazione del codice:

* Il codice Java dev’essere indentato in maniera appropriata (ese: il checkstyle di Google)
* Il codice HTLM dev’essere indentato in maniera appropriata (ese: il checkstyle di Google)
  1. **Definizione, acronimi e abbreviazioni**

CheckStyle: strumento di analisi statica del codice utilizzato nello sviluppo software per verificare se il codice sorgente Java è conforme alle regole di codifica specificate.

* 1. **Riferimenti**
* Design goals: sezione 1.2 dell’SDD
* Scelta dell’ambiente d’esecuzione: sezione 3.2 e 3.3 dell’SDD
* Spring: “Inserie collegamento Spring”
* Ionic: “Inserire collegamento a Ionic”

1. **Packages**

In questa sezione presentiamo in modo più approfondito quella che è la divisione in sottosistemi e l’organizzazione del codice in file

* 1. **Divisione in pacchetti**

La divisione del sistema (applicazione desktop) condivide molti aspetti con quella proposta nel documento di System Design: il sistema e diviso in 4 pacchetti presentazione, business, persistence e entity.

Il pacchetto presentazione contiene tutte le classi per la generazione dell’interfaccia, il pacchetto business contiene le classi per quanto riguarda le operazione che potra effettuare l’utente, il pacchetto persistence contiene la classe contenete tutte le operazioni possibili sul database e il pacchetto entity conterra le classi che rappresenteranno l’entita del database.

Inserire divisione in pacchetti web app.